CODING AND TRANSFORMATION OF INFORMATION(Dissertation_全文)

AUTHOR(S):

Nishio, Hidenosuke

CODING AND TRANSFORMATION OF INFORMATION

by

HIDENOSUKE NISHIO

DEPARTMENT OF ELECTRICAL ENGINEERING

KYOTO UNIVERSITY

DECEMBER, 1965

CODING AND TRANSFORMATION OF INFORMATION

by

HIDENOSUKE NISHIO

DEPARTMENT OF ELECTRICAL ENGINEERING

KYOTO UNIVERSITY

DECEMBER 1965

PREFACE

Since about twenty years, the science of information has appeared and grown up. In the early years it was called the information theory and concerned mainly with the probabilistic phenomena arising in various fields of the science. The information theory, which was founded by C.Shannon, was the mathematical formulation of the electric communication system. The electric communication had been investigated by engineers from the practical point of view before then, but this theory gave them insight into the whole system. One of the most eminent contributions of the theory is that it established the concepts of the information measure and of the encoding of information. The original information can not be transmitted effectively unless it is encoded into the appropriate code. The theory of codes then appeared. Its importance could be compared with that of the modulation and transmission technique.

A little after the information theory, the field of information processing appeared with rapid progress of the digital computer. Here information is not regarded as the probabilistic object but is treated as the deterministic one. Information in this field has the form of the symbol such as the numeral or the alphabetical letter. The single symbol is almost meaningless and what is interesting is the system of symbols. The language,

mathematics, the system of codes and so on are examples of such systems. Though the numerical computation is often considered to be the representative of the information processing by the computer, the essence of this field may be the transformation of systems of symbols. The mechanical translation of languages, the compiler system of programs, etc. are typical examples. Theory of algorithms, theory of logical functions and theory of logical circuits are related fields. Recently the study of logical machines, which help the human being in the logical work, has stimulated interest of the investigator. The automatic design of machines and the theorem proving of mathematics are such examples.

In studying the science of information one should treat the above mentioned both fields together, because information has two sides.

One of the typical examples, in which these both phases of information meet each other, is the pattern recognition. In general the pattern recognition should be regarded as one kind of statistical communication, but the information theory itself does not give useful tools for investigating this complex field. The other theory has been expected in vain. The second phase of the science of information seems to be useful for this purpose, if it is modified. In addition to it, something like the learning mechanism would be necessary. The general theory of pattern recognition is the field to come.

In this volume the works, which the author finished during the post-graduate course, are presented.

The volume is divided into two parts, which treat the problems of coding of information and the foundation of the transformation theory of program respectively.

PART I is devoted to coding problems of information. In CHAPTER II the study on the minimum distance code is presented as well as the mathematical formulation of codes in general. The algorithm to generate minimum distance codes was established and some interesting results were obtained by using the computer. In CHAPTER III the problem of coding patterns into logical functions is treated by means of the trial and error model. The probabilistic logic circuit is introduced there. CHAPTER IV is an application of the group code theory to approximation of logical functions. This is an answer to the problem arising at the trial and error model.

PART II is devoted to translation of the program into the logical circuit. It consists of four chapters. CHAPTER I is introduction and CHAPTER II is formulation of the problem. CHAPTER III is description of the logical machine and its programming system, which we chose for the study of transformation. In CHAPTER IV the algorithm of transformation is given as well as some examples.

November, 1965

Hidenosuke Nishio

ACKNOWLEDGEMENT

The author wishes to express his hearty thanks to Professor Toshiyuki Sakai, who guided him to this field, for his constant support, understanding and encouragement during the study of this work. He also wishes to thank Professor Ken-ichi Maeda, who was the instructor of the author at the early period of the study. Thanks are due to Dr. Shuzo Yajima, Mr. Shuji Doshita and Mr. Makoto Nagao for general interest, discussion and suggestion on the information processing, the digital computer and the pattern recognition. The author is greately indebted to Mr. Shigeyasu Kitawaki for cooperation and critisism on the theory of codes and for his private correspondence about the theorem of CHAPTER III, and to Mr. Koichi Tabata for his cooperation and help in the study of the trial and error model. Messrs. Yasunaga Niimi, Ichiro Matsuo and Kentaro Soga made useful discussions with the author on the theory of logical functions and mathematics. Messrs. Sin-ichi Murai, Masanori Amano, Toshiaki Inui and Hidenori Kawakami helped the author in the study of PART II. Staffs of the computation center of Kyoto University helped him in utilizing the computer KDC-I. Author's thanks are due to Miss Minako Matsumoto, who typed the whole manuscript.

CONTENTS

PART II . TRANSFORMATION OF PROGRAM INTO LOGICAL CIRCUIT

PART I

CODING PROBLEMS OF INFORMATION

CHAPTER I

Introduction

Since in 1948 C.Shannon formulated the mathematical theory of information, a great deal scientists and engineers have devoted their contributions to the foundation and the application of the information theory. This field treats the communication from the statistical point of view. Among other subjects concerning the transmission, the reception and the reproduction of signals, the coding of information is the main part of the theory. Shannon's original coding theorem guarantees the possibility of efficient communication.

On the other hand the theory of codes has been investigated since about 1950 R.Hamming established the concept of error-correcting codes. This treats the methods to construct actually the efficient codes which realize the ideal communication.

Another phase of the information theory is its application to the processing of information. The typical examples of the information processing are the digital computer and the pattern recognition. This field has developed independently from the information theory. Its complete theory has not yet been established, though experimentally and practically there was great success in constructing information processing machines.

The following three chapters are devoted to the second and the third phases of the study of information.

Here we review shortly the history and the interrelation of both fields.

Though we do not treat Shannon's theory itself in this thesis, we can not do without referring to his main result "the coding theorem" in order to introduce ourselves into the subject smoothly. Among his contribution as well as those by other authors, the coding theorem is most important and valuable in the sense that it indicates the possibility and limitation of the coding of information. The theorem states that there is a coding of the information source such that the error probability of decoding is as small as desired if the transmission rate is less than the channel capacity. The converse theorem was also proved. Note that the theorem holds only when the channel is fixed. It is the main problem in applications of the theorem to determine the channel itself.

Unfortunately, however, these theorems do not suggest directly the method to construct the adequate code for the ideal transmission of information. It is only known from the proof of the theorem that very long code is seemingly required to reach the Shannon-bound. Therefore the study of codes was required. The theory of codes began with the excellent paper by H.W.Hamming in

1950[1]. He founded the concept of the error correcting and the error detecting codes. The Hamming distance has played an important role in this theory, as well as in many practical fields of the information processing. The minimum distance code was investigated much since then. The second milestone will be the group code, which was found by D.Slepian and Z.Kiyasu about 1956. The former used the computer to obtain the optimal codes[8]. Today it is quite usual to employ the computer for the study of codes. The mathematical structure of group codes is appropriate for the computation. But the computation arising in the theory of codes is met with such difficulties as in the combinatorial problems. The computation time increases in the factorial order of the code length. It is therefore the ordinary methodology to find the theory or the algorithm which reduces the speed of increas of the computation time, besides to get the theory to construct more economical codes.

The third big event was the burst error correcting code by Bose and Chaudhuri in 1960[40]. The linear switching circuit was applied cleverly to the encoding and the decoding of the cyclic code by W.W.Peterson[41]. One of the features of these codes is that the relatively long code can be constructed easily and the encoder and the decoder can be also easily implemented, which is important from the engineer's point of view. Today the

large scale data transmission is being investigated using these techniques as well as the modern modulation technique.

Since the end of the last decade, the digital technique, of which the computer is typical, has been so developed that a great deal digital data may be processed in a practical time delay and with reasonable expense. This offers effective terminal equipments of the transmission system. This kind of cooperation between the theory of codes and the digital technique is interesting.

Contrary to the theory of codes, the field of information processing like the pattern recognition has been not yet cultivated sufficiently. Though it seems very much that the information theory and the theory of codes can be applied to this field, there is no definite theory for it. One reason for it might be the fact that the information theory is based on the law of large numbers in the probability theory. For the pattern recognition theory, this concept can not be applied directly. In the pattern recognition the concept of statistical decision making is essential and the investigation of the information source itself is also important. On the other hand the study of logical functions is necessary because the recognition algorithm is expressed in terms of them often. The mathematical formulations of the code and the logical function have one to one correspondence. There-

fore there is a possibility of applying the theory of code to the pattern recognition.

The pattern recognition can be regarded as a coding problem of information. But contrary to ordinary problems, where the information source is the set of abstract symbols with probabilities of appearance, we should consider at first the measurement of concrete patterns. It is decisive to the system, what is chosen as the observed entity. If we choose the ordinary $n \times m$ rectangular mesh for the observation of the printed character, each observed image corresponds to a nm-tuple of 0 and 1. The ordinary pattern recognizers process this form of information by means of the logical circuit. In general the pattern will appear on the mesh in disturbed form, sometimes shifted and sometimes affected by the noise. Therefore a pattern is considered to be a set of nm-tuples or a binary code of length nm. Therefore the pattern recognition can be said to indicate, to what code the observed nm-tuple belongs. The design of the machine is to design the code for the pattern. In this case there is essential difference between the ordinary coding and the coding of pattern. At the former the form of the code is determined so that its transmission may be affected little by the noise and the efficiency (transmission rate) may be as high as possible. In this theory only the probability of the information source and the

nature of the transmission channel are considered. Shannon's theorem gives the hopeful result under these conditions.

In the case of the pattern recognition, however, the coding problem is so complicated that it is hardly formulated well. The main difficulty arises in connection with the decision criterion of man. Any concrete pattern is recognized by man at first. We can not therefore design the code only from the theoretical point of view. The structure of codes should represent such features as the geometrical and the topological configurations, which are seemingly essential to the pattern. Furthermore for encoding patterns, measurement of the concrete patterns decides the system of codes. If the measurement is not adequate, effective coding can not be expected. In the mathematical statistics the theory of measurement has been investigated from the information theoretical point of view. This, however, does not serve design of the pattern recognition system directly.

One approach to the design of measurement will be as follows: At first we choose some codes, which have good features as the code in some sense. Then the code is tested by experiments, if it is suitable for the pattern recognition with specific measuring ststem. If not, the measuring system is modified, until finally the code is determined to be accepted or not.

This approach may seem contradictory. Because the design of codes begins with investigation of the transmission channel, while in our case there is not what corresponds precisely to the channel. But it will give some suggestions on the theoretical approach to the measurement to utilize results of the theory of codes.

In this sense, too, besides in the original sense, the error-correcting and -detecting code is the interesting subject of the study. Among many codes, we treat in CHAPTER II the minimum distance codes. The minimum distance code is mathematically a subset of the set of elements, for which any pair of elements are separated at least by a certain distance. In the case of application the distance corresponds to the ability of discrimination of objects. We persue the codes which contain the maximum number of elements.

Quite independently from the minimum distance code, there is a subset, for which any pair of elements are separated at most by a certain distance. It is called $\mathcal{E}$-net and used in the theory of approximation of functions. The minimum sets are persued in this case.

In CHAPTER IV a theory is given which indicates the interrelation between the $\mathcal{E}$-net and the error correcting code. This theory was developed in connection with the trial and error model of coding of patterns, which will appear in CHAPTER III.

CHAPTER II

Theory of Codes

§ 2.1 Introduction

As mentioned in CHAP.I, the theory of codes is doubly involved in the study of the information processing. At first the former is expected to serve the latter, because the coding problem is the starting point of the information processing in general. Secondly the former has been developed thanks to the latter, in particular to the digital computer.

Hitherto the second cooperation has been pursued much more than the first one. This is because the study of the information processing has been done experimentally and could not be treated theoretically so easily. For example the pattern recognition has been investigated by constructing the machine or by simulating the behavior of the conceptual machine. One reason for this is that the information processing should be always of use in practice and the useful theory has not been established, though in the case of rather simple systems such as the code conversion, the data transmission and so on, the theory of transmission of information and the theory of codes seems to be useful.

Contrary to this, the first cooperation has been made by a great deal theoreticians. One of the most fruitful

results of the computer use was the search of the optimal group codes by D.Slepian in 1956[8]. He formulated the theory of the error correcting systematic codes using the group theory and calculated out many interesting codes with IBM 704 which was the biggest computer at that time. About 1960 W.W.Peterson[54] used the computer to find the best cyclic codes of relatively large length.

In general the theory of codes is reduced to the study of a type of the combinatorial problem. At first a theory is established by investigating the mathematical structure of the considered system of codes, and then an algorithm, which will give the desired codes as efficiently as possible, is written, and finally the computer is used to obtain them actually. Therefore it would be a success to the investigator, if he finds a more efficient algorithm.

It is said however that it would be impossible to get the general algorithm which permits us to calculate the optimal code for any given length and number of information bits. The programs obtained so far are limited to a certain region of the length, the number of the information bits and the number of the errors which can be corrected.

In the case of the non-group codes such as the minimum distance code, it is more difficult to use the computer for finding the best codes because of the lack of

an adequate methematical structure. It takes much time and memory to look for the best codes by examining all possibilities one after another. N.Wax[14] used the computer cleverly in calculating the volume of some geometrical objects to get the best code.

In § 2.2 the mathematical prerequisites for the theory of codes is presented and in § 2.3 the algorithm called Standard Process is described. It gives all minimum distance codes with the specified length and the specified minimum distance. The computational results of the algorithm are shown, among which there are some interesting codes. Then an interesting and important theorem about S.P. and the group code is given, as well as some computational results concerning with the maximum minimum distance group codes.

§ 2.2 Mathematical formulations of codes [8, 52, 54]

Some mathematical tools for the study of codes are listed in this section.

§ 2.2.1 Set of the n-tuples X^n

The code is a set of objects which are called code words or elements of the code. In this thesis the binary block code is investigated exclusively. Each code word is therefore a n-tuple of 0 and 1, where n is a positive integer and called the length of code. The set of all n-tuples is denoted by X^n or simply by X. We treat the communication model, for which the set of the transmitted signals and that of the received signals are identical and X^n. The code of length n is a subset of X^n and denoted by C(n). The code word x of C(n) is represented by a vector $(x_1, x_2, \ldots, x_n)$, where x_i is either 0 or 1 and called the (i-th) component of the code word x. When a concrete code word is referred to, the notation such as 00001111 is used. A concrete code is represented by the array as shown in Fig. 2.1.

$$C(8) = \begin{array}{l} 00000000 \\ 00001111 \\ 00110011 \\ 00111100 \end{array}$$

Fig. 2.1

Of the array the row is the code word and the column

indicates the place of the component.

The number of code words contained by the code C(n) is denoted by [C(n)].

Two codes C_1 and C_2 are identical if they consist of the same elements. Two codes C_1 and C_2 are said to be equivalent if C_1 is obtained by permuting the columns of C_2. This is because they have the same effect as the code, if the transmission channel is the binary symmetric channel without memory. As shown later the behavior of the code against the noise is determined by its metric structure. In this respect C_2 is equivalent to C_1 if C_2 is obtained by changing all components of C_1 from 0 to 1 and vise versa.

Decision scheme

The decision (decoding) scheme of C(n) is a partition of X^n into subsets such that every element of C(n) belongs to one and only one subset, where the partition of a set is its classification into some subsets so that every element belongs to one and only one subset. A received code word is said to be decoded, if it is determined to which subset it belongs. The receiver decides then that the code word, which corresponds to the said subset has been transmitted.

In the practical communication channel the noise disturbs the transmitted signal so that the received code word is not the same as the transmitted one. In

this case the noise is said to cause the error.

We agree to say " An error occurred at the i-th component ", if the i-th component of the received code word does not equal that of the transmitted one. " t errors occurred ", if t components are different in the transmitted and received code words ($0 \leq t \leq n$).

A vector (e_1, e_2, ..., e_n) is called the error pattern if it is the difference of the transmitted and the received code words. The number of all error patterns of k errors is ${}_nC_k$.

For the binary symmetric channel the probability that a specified error pattern of k errors occurs is $p^k(1-p)^{n-k}$, where p is the probability of incorrect decision of each component. The probability that k errors occur is ${}_nC_k p^k(1-p)^{n-k}$.

A error pattern is said to be corrected or correctable if the received code word suffering it is decoded by the decision scheme as belonging to the subset which corresponds to the transmitted code word. A code is called the t-error correcting code if there is a decision scheme such that all error patterns of errors $\leq t$ can be corrected. In general the t-error correcting code corrects some other error patterns of errors $> t$ than the said errors.

A partition of X^n is called the error detecting decision scheme of C(n) if the number of the subsets

is $[C(n)]+1$ and each code word belongs to one and only one subset. In the same way as the error correcting scheme the error detecting scheme can be used to "correct" errors. An error pattern is detected if the received code word suffering it is found in the subset which does not contain any code word.

In this thesis the error detection is not treated, though this technique is being investigated by many authors with respect to the data transmission.

Metric d

Since in 1950 Hamming[1] indicated that the "Hamming distance" is closely related to the error correcting or detecting ability of the code, the metric structure of the code has been investigated.

For two elements x and y of X^n the Hamming distance $d(x,y)$ is the number of components which are different in both elements. The Hamming distance satisfies three axioms of the metric. Therefore X is considered to be a metric space (X, d).

Definition 1. A code C is called to have the minimum distance d and denoted by $C(n,d)$, if the following relations hold;

(i) $d(x,y) \geq d$, for any distinct x and y.

(ii) $d(x,y) = d$, for some x and y.

(iii) (i) and (ii) do not hold for d-1 in place of d.

Definition 2. For fixed n and d, a $C(n,d)$ is

called the maximum minimum distance code (MMDC) and denoted by M(n,d), if it contains the greatest number of elements among all C(n,d)s.

Order

X^n is considered to be the totally ordered set, if the order is defined by that of the binary number. The element $(x_1, x_2, \ldots, x_n)$ can be identified with the number $x_i 2^{n-i}$. In X^n, 000...0 is the smallest element, while 111...1 is the largest one. In § 2.3 this order is used for the computer algorithm "Standard Process".

§ 2.2.2 Operations

Since in 1956 D.Slepian[8] pointed out that the theory of groups is useful for designing the error correcting code and its dicision scheme, the code which is called the group code has been studied by many investigators and found to be a powerful code with respect to its error control ability and its encoding and decoding procedure which can be easily implemented.

Almost all features of this code come from its mathematical structure, in particular its algebraic structure.

In the following some definitions about it are given.

Group G

A group G is a set of elements, for which a binary operation + is defined and the following axioms G1 to G4 hold.

G1. for any x and y in G, $x+y \in G$.

G2. for any x, y and z in G, $(x+y)+z = x+(y+z)$.

G3. There is an identity element 0 such that for any x in G, $x+0 = 0+x = x$.

G4. For any element x in G, there is an inverse element $-x$ such that $x+(-x) = (-x)+x = 0$.

The set of all n-tuples X^n is a group with the operation defined as follows:

If $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$, then $x+y = (x_1+y_1, x_2+y_2, \ldots, x_n+y_n)$, where $+$ is the modulo two sum. The identity element is $(0, 0, \ldots, 0)$ and the inverse element of x is x.

A group is called the Abelian group if $x+y = y+x$ for any $x, y \in G$.

Rings R

A ring R is a set of elements, for which two binary operations are defined, say $+$ and $\cdot$, (the second operation symbol $\cdot$ is sometimes omitted for the sake of simplicity, i.e. $x \cdot y \equiv xy$.) and the following four axioms are satisfied.

R1. R is an Abelian group under the operation $+$.

R2. For any x and y in G, $xy \in G$.

R3. For any x, y and z in G, $x(yz) = (xy)z$.

R4. For any x, y and z in G, $x(y+z) = xy+xz$ and $(x+y)z = xz+yz$.

A ring is called commutative, if $xy = yx$ for any

x and y.

Fields F

A field F is a commutative ring which has a multiplication identity elements e such that for any $x \in G$, $ex = xe = x$, and every nonzero element x has an inverse element x^{-1} such that $xx^{-1} = x^{-1}x = e$.

Vector space V

A set of elements V is called a vector space or a linear space over a fied F, if it satisfies the following five axioms.

V1. V is an Abelian group under operation +.

V2. For any v in V and c in F, a product cv is defined and $cv \in V$.

V3. For any u and v in V and c in F, $c(u+v) = cu+cv$.

V4. For any v in V and c and d in F, $(c+d)v = cv+dv$.

V5. For any v in V and c and d in F, $(cd)v = c(dv)$ and $ev = v$.

The set $\{0, 1\}$ consisting of 0 and 1 is a field with the modulo two sum and the ordinary multiplication.

The set X^n is a vector space over $\{0, 1\}$, where the operations are defined as follows:

If $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$, then $xy = (x_1y_1, x_2y_2, \ldots, x_ny_n)$ where the product x_iy_i is the ordinary multiplication of numbers. If $c \in F$,

then $cx = (cx_1, cx_2, \ldots, cx_n)$ where the product cx_i is the ordinary one.

In this thesis X^n is considered to be the vector space over $\{0, 1\}$, and denoted by V^n sometimes.

Subgroup

A subset H of a group G is called the subgroup of G if it is also a group.

Coset and coset leader

Given a finite group G and its subgroup H, all elements of G can be arranged in the form of the array as shown in Fig.2.2, where $G = \{g_1, g_2, \ldots, g_{mn}\}$ and $H = \{h_1, h_2, \ldots, h_n\}$. The first row is H where $h_1 = 0$. In order to construct the second row, g_2 is chosen from G-H. (The symbol - indicates the difference of sets in the set theory.) The 2nd row i-th column element is the sum g_2+h_i. The i-th row is made by choosing an element g_i from $G-(g_2+H)-(g_3+H)- \ldots -(g_{i-1}+H)$. The procedure proceeds until all elements appear somewhere in the array. Each row (g_i+H) is called a (left) coset of H and g_i is called its coset leader.

If the coset are constructed by the operation $H+g_i$ in stead of g_i+H, then they are called right cosets. If G is an Abelian group, the left and right cosets are identical and called cosets simply.

$$
\begin{array}{lccccccc}
g_1 = h_1 & h_2 & h_3 & \cdot & \cdot & h_i & \cdot & h_n \\
g_2 & g_2+h_2 & \cdot & \cdot & & g_2+h_i & & g_2+h_n \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
g_j & g_j+h_2 & \cdot & \cdot & & g_j+h_i & & g_j+h_n \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
g_m & g_m+h_2 & \cdot & \cdot & \cdot & \cdot & & g_m+h_n
\end{array}
$$

Fig. 2.2

The following well known theorems are used in the theory of codes.

P1. Two elements x and y are in the same coset of H if and only if $x-y \in H$.

P2. Every element of G is in one and only one coset of H.

P3. Any element of a coset can be its coset leader, i.e. if $g \in g_i+H$, then $g+H = g_i+H$.

Group codes

A subgroup of V^n is called a group code and denoted by $G(n)$. If the dimension of $G(n)$ in the space V^n is k, then the code is denoted by $G(n, k)$. When $G(n, k)$ is used with the error correcting decision scheme, k is equal to the number of the information bits.

If $G(n)$ has the minimun distance d, it is denoted by $G(n, d)$, and called the minimum distance group code.

Weight

The (Hamming) weight of a vector x is the number of ones contained in it and denoted by $\| x \|$. Then

$$d(x, y) = \| x - y \| \qquad (2.1)$$

Therefore the metric structure of the group code is completely determined by the weight of each element.

§ 2.2.3 Decision scheme and the error control ability of the code

In the case of the binary symmetric channel, the optimal code is the code $G(n,k)$ which has the decision scheme such that the average probability of incorrect decoding $P(e)$ is the smallest among all $G(n,k)$s. The problem persued is to search the optimal codes for n and k. In this respect the decision scheme is investigated further.

Let X $(=X^n)$ and Y $(=X^n)$ be the sets of all n-tuples of the transmitted and the received code words. Let $p(y/x)$ be the conditional probability that y is received when x was transmitted. That is $(X, Y, p(\ /x))$ is a discrete channel without memory in the Shannon's formulation.

Let $\{ A_1, A_2, \ldots, A_N \}$ be the decision scheme of the code $C(n) = \{x_1, x_2, \ldots, x_N\}$ where $N = [C(n)]$. And we agree that whenever the received element is found in A_j, we assume that x_j was transmitted. Then A_i con-

tains x_i by definition. If $p(x)$ is the probability that x is transmitted, $p(y) = \sum_{x \in X} p(x)p(y/x)$ is the probability that y is received. Then $P(e)$ is given by (2.2).

$$P(e) = \sum_{y \in Y} p(y)[\, 1 - p(x_y/y)\,] \qquad (2.2)$$

where $x_y = x_i$ if $y \in A_i$.

In the case of the binary symmetric channel, $P(e)$ is a function only of $d(x,y)$ and p (the error probability of each component). In fact

$$p(y/x) = p^{d(x,y)}(1-p)^{n-d(x,y)} \qquad (2.3)$$

$p(y/x)$ is a monotonically decreasing function of $d(x,y)$ since $1-p > p$.

In particular, if the decision scheme corrects all error patterns of errors $\leq t$,

$$P(e) \leq 1 - \sum_{i=0}^{t} {}_nC_i p^i (1-p)^{n-i} \qquad (2.4)$$

In order to see that the code which corrects many errors is not always good with respect to $P(e)$, compare the following two codes, $C(7,3)$ and $C(10,5)$, for which $[C(7,3)] = 16$ and $[C(10,5)] = 12$. If we consider the error pattern of one error for $C(7,3)$ and one and two errors for $C(10,5)$ only and denote the error probabilities by $P_1(e)$ and $P_2(e)$, the region of p for which $P_1(e) > P_2(e)$ is $0 < p < 0.435$. That is, if the channel is too noisy it is not preferable to use the 2-error

correcting code.

In the case of group codes the decision scheme is determined using the cosets. (See, Fig. 2.2). For each coset the coset leader is chosen so that the weight of it is the smallest among all elements of the same coset. Then a received code word is decoded as the code word which is written at the top of the column to which it belongs. With this scheme the error patterns which are identical with the coset leaders can be corrected.

The t-error correcting code has the scheme for which all elements of the weight $\leq t$ are coset leaders.

A group code that for some m has all error patterns of weight m or less and no others as coset leaders is called a perfect code. A code which for some m has all patterns of weight m or less, some of weight $m+1$, and none of greater weight as coset leaders is called quasi-perfect. For the binary symmetric channel it is known the perfect and the quasi-perfect codes are optimal.

The following proposition connects the minimum distance and the error correcting ability of a code.

The code $C(n)$ has the decision scheme which permits the t-error correction, if and only if its minimum distance $d \geq 2t+1$.

§ 2.3 Maximum Minimum Distance Codes

The study of the maximum minimum distance code (MMDC) is one of many approaches to the optimal code or the effective error correcting code. As mentioned in the previous sections the quality of a code could not be judged only from its metric structure but the feasibility of the decoding scheme should be taken into account. In this respect MMDC is not always suitable since the decoding scheme is not simple. But theoretically it suggests much about the rate of the information bits to the whole length of the code.

The problems concerning MMDC are formulated as follows: To find the method to calculate $[M(n,d)]$ for any n and d and to obtain $M(n,d)$s themselves.

§ 2.3.1 Upper bounds for $[M(n,d)]$ [1, 14, 16, 20~34]

These problems have not been solved completely. The contributions by many authors gave the answer for small n and d and for some special combinations of n and d. One of the methods was to get the good upper bounds for $[M(n,d)]$ as well as the lower bounds. In the following are listed several equalities and inequalities about the function $[M(n,d)]$ of n and d.

(i) $[M(n,d)]$ is the function of n ($n \geq 1$) and d ($n \geq d \geq 1$) whose value is a positive integer $\leq 2^n$.

(ii) $[M(d,d)] = 2$, $[M(n,1)] = 2^n$

$$[M(n,2d-1)] = [M(n+1,2d)]$$

(iii) $[M(n,d)] \leq [M(n+1,d)]$, $[M(n,d)] \geq [M(n,d+1)]$

(iv) $[M(n,d)] \leq 2^n/(1+{}_nC_1+{}_nC_2+\ldots+{}_nC_t)$, where

$t = [\ d-1/2\ ]$ (Gauss' symbol) (by Hamming)

(v) $[M(n,d)] \leq 2^{n-d+1}$ (by Joshi)

These functional relations, even together with others found by many investigators, are not sufficient to solve $[M(n,d)]$ generally. They serve the partial solution and the upper and the lower bounds of $[M(n,d)]$. Many values of $[M(n,d)]$ for small n and d were obtained by getting the gap between two bounds as narrow as possible. The right most entries of Table 2.1 are the best upper bounds of $[M(n,d)]$ which were obtained using mainly the results of Wax and those of Plotkin. The middle entries are $M(n,d)$s which were actually obtained by several authors.

§ 2.3.2 Standard Process (S.P.)[42, 43]

Contrary to the above mentioned mathematical approach we formulated the algorithm to generate $C(n,d)$ in general which is expected to give some suggestion about $[M(n,d)]$. The algorithm, called "Standard Process (S.P.)", generates all $C(n,d)$s for given n and d.

Standard Process

Let X^n be the ordered set with the order of binary number (§2.2), so that any subsets of X^n has the small-

est element denoted by min S.

Step 1: Construct the sequence of subsets $S_0, S_1, \ldots, S_k$ of X^n in turn as follows:

$$S_0 = X^n$$

$$S_1 = \{a: d(a_o,a) \geq d, \quad a \in S_0\} \neq \emptyset, \text{ where } a_o = \min S_0$$

$$S_2 = \{a: d(a_1,a) \geq d, \quad a \in S_1\} \neq \emptyset, \text{ where } a_1 = \min S_1$$

$$- - - - -$$

$$S_{i+1} = \{a: d(a_i,a) \geq d, \quad a \in S_i\} \neq \emptyset, \text{ where } a_i = \min S_i$$

$$- - - - -$$

$$S_k = \{a: d(a_{k-1},a) \geq d, \ a \in S_{k-1}\} = \emptyset,$$

where $a_{k-1} = \min S_{k-1}$

In the equations above, $\emptyset$ stands for the empty set and $\{a: \quad\}$ for such a set that satisfies the condition stated after the symbol :.

Since X^n is finite, it is clear that there exists a positive integer k with the above mentioned property. If k=1, there is no C(n,d) for that n and d. This may be the case when $n < d$. It is easily seen from the construction of S_i that the subset $\{a_0, a_1, \ldots, a_{k-1}\}$ is a C(n,d). This code is denoted by $C_1(n,d)$.

Step 2: If in the sequence of subsets $S_0, S_1, \ldots, S_{k-1}$ constructed by Step 1 there is S_i such that

$$S_i \cap \overline{S_{i+1}} - \{a_i\} \neq \emptyset \qquad (2.5)$$

and i is the largerst number which satisfies the condition (2.5), then construct another sequence of subsets as follows:

$$S'_j = S_j \quad \text{for} \quad j = 0, 1, \ldots, i$$

$$S'_{i+1} = \left\{ a: d(b_i, a) \geq d, \quad a \in S'_i \right\} \neq \emptyset,$$

where $b_i = \min S_i \cap \overline{S_{i+1}} - \{a_i\}$

$$S'_{i+2} = \left\{ a: d(b_{i+1}, a) \geq d, \quad a \in S'_{i+1} \right\} \neq \emptyset,$$

where $b_{i+1} = \min S'_{i+1}$

- - - - - - -

$$S'_{k'} = \left\{ a: d(b_{k'-1}, a) \geq d, \quad a \in S'_{k'-1} \right\} = 0,$$

where $b_{k'-1} = \min S'_{k'-1}$

In general the new sequence does not equal the old one and $k' \neq k$. If we use the new symbolism b_j in place of a_j for $j = 0, 1, \ldots, i-1$, it is easy to see that the new sequence $b_0, b_1, \ldots, b_{k'-1}$ is another $C(n,d)$, which is denoted by $C_2(n,d)$.

If there is no positive integer satisfying the condition (2.5) there is only one minimum distance code $C_1(n,d)$ for n and d. This may be the case, for instance, when $d = n$, and $C_1(n,d)$ consists of two elements.

Step 3: In the same manner as Step 2 we construct the third sequence of subsets, if there is positive integer satisfying the following condition;

$$S'_j \cap \overline{S'_{j+1}} - \{b_j\} \neq \emptyset \qquad (2.6)$$

and if $j = i$,

$$S_i' \cap \overline{S_{i+1}'} - \{b_i\} - \{a_i\} \neq \emptyset \qquad (2.7)$$

Let j be the maximal number which satisfies the conditions (2.6) and (2.7). The new sequence is constructed as follows:

$$S_n'' = S_n' \quad \text{for} \quad n = 0, 1, \ldots, j$$

$$S_{j+1}'' = \{ a: d(c_j, a) \geq d, \quad a \in S_j' \} \neq \emptyset,$$

$$\text{where} \quad c_j = \min S_j'' \cap \overline{S_{j+1}''} - \{b_j\} - \{a_i\}$$

- - - - - - -

$$S_{k''}'' = \{ a: d(c_{k''-1}, a) \geq d, \quad a \in S_{k''-1}'' \} = \emptyset,$$

$$\text{where} \quad c_{k''-1} = \min S_{k''-1}''$$

If we write $c_n = b_n$ for $n = 0, 1, \ldots, j-1$, then the subset $\{c_0, c_1, \ldots, c_{k''-1}\}$ is the third $C(n,d)$ and denoted by $C_3(n,d)$. If there is no positive integer which satisfies the conditions (2.6) and (2.7), $C_1(n,d)$ and $C_2(n,d)$ are the only two minimum distance codes.

Step m: After m codes have been obtained in this manner, S.P. is concluded if there is no positive integer which satisfies the condition (2.8).

$$S_i^m \cap \overline{S_{i+1}^m} - \{a_i\} - \{b_i\} - \{c_i\} - \ldots \neq \emptyset,$$
$$\text{for some} \quad i. \qquad (2.8)$$

Then $C_1(n,d)$, C_2, ..., C_m are the whole set of minimum distance codes for given n and d.

As is seen from the algorithm, all C(n,d)s are generated by S.P. but it is not guaranteed that all generated codes are different or not equivalent. It is impossible to estimate the number m for n and d, but it is supposed to be a very large number.

Results of computation

The Standard Process can be easily programmed for the computer. We used the medium scale computer KDC-I, which was installed at Kyoto University in 1960. Like other combinatorial problems S.P. requires a great deal of memory spaces and high speed computation. With KDC-I, which has the memory of 4200 digit words, even the case for n=9 and d=3 could not be computed. Among programming techniques necessary for the S.P., there is the efficient evaluation of the Hamming distance of two code words x and y, d(x,y) or $\|x-y\|$, which takes much computation time if the instruction system of the computer is not suitable. The operation of taking the weight of a binary word is found often in the computer use related to the theory of codes and the theory of logical functions. Since the present computers do not have the special instruction for it, the routine using the shift and the count operations is necessary.

Table 2.1 shows the results of the computation. The left most entries are [M(n,d)]s generated by S.P. and [C(n,d)]s which do not reach the maximality. The other

n \ d	3			5			7		
6	8	8	8	2	2	2			
7	16	16	16	2	2	2			
8	18	20	20	4	4	4			
9	-	38	39	6	6	6			
10	-	68	82	12	12	12			
11		128	154	24	24	24	4	4	4
12				26	32	46	4	4	4
13				48	48	85	8	8	8
14							16	16	16
15							32	32	32
16							32		
17							64		

The left most entries are the results of S.P.. The middle ones are the values actually obtained by other authors. The right most ones are the upper bounds given by Wax and Plotkin.

Table 2.1 [M(n,d)]

entries are given for the purpose of comparison. Fig. 2.3 shows some concrete codes generated by S.P., with exceptions M(10,5) and M(11,5) which were obtained by modifying the process. S.P. was found not efficient to generate M(n,d) fast and modified. At an arbitrary step, the next element is taken not necessarily as the smallest element of the subset S_i. The algorithm of modification has not been established and it is determined by intuition, at which step and how the standard process is to be modified during the computation. Such

$C_1(6,3) = M(6,3)$

$[C_1(6,3)] = 8$

000000
111000
100110
011110
010101
101101
110011
001011

$C_2(6,3) \neq M(6,3)$

$[C_2(6,3)] = 6$

000000
111000
100110
011110
010101
101011

$C_1(7,3) = M(7,3)$

$[C_1(7,3)] = 16$

0000000
1110000
1001100
0111100
0101010
1011010
1100110
0010110
1101001
0011001
0100101
1010101
1000011
0110011
0001111
1111111

$C_2(7,3) \neq M(7,3)$

$[C_2(7,3)] = 13$

0000000
1110000
1001100
0111100
0101010
1011010
1100110
0010101
1101001
1000011
0110011
0001111
1111111

$C_1(8,3) = M(8,3)$

$[C_1(8,3)] = 16$

00000000
11100000
10011000
01111000
01010100
10110100
11001100
00101100
11010010
00110010
01001010
10101010
10000110
01100110
00011110
11111110

$C_1(8,5) = M(8,5)$

$[C_1(8,5)] = 4$

00000000
11111000
11000111
00111111

$C_2(8,5) = M(8,5)$

$[C_2(8,5)] = 4$

00000000
11111000
10100111
01011111

$C_1(9,5) \neq M(9,5)$

$[C_1(9,5)] = 4$

000000000
111110000
110001110
001111110

$C_2(9,5) = M(9,5)$

$[C_2(9,5)] = 6$

000000000
111110000
110001110
001101101
101011011
010110111

Fig. 2.3 Codes generated by S.P.

$C_1(10,5) \neq M(10,5)$

$[C_1(10,5)] = 8$

0000000000
1111100000
1100011100
0011111100
1010010011
0101110011
0110001111
1001101111

$C_2(10,5) \neq M(10,5)$

$[C_2(10,5)] = 8$

0000000000
1111100000
1100011100
0011111100
1010010011
0101110011
0001001111
1110101111

$C_i(10,5) = M(10,5)$

$[M(10,5)] = 12$

0000000000
1111100000
1100011100
0011011010
1010110110
0101101110
0110111001
0001110101
1011001101
1101010011
1000101011
0110000111

(i is unknown.)

$C_j(11,5) = M(11,5)$

$[M(11,5)] = 24$

00000000000
11111000000
11000111000
00110110100
10101101100
01011011100
01101110010
00011101010
10110011010
11010100110
10001010110
01100001110
10011110001
01110101001
00101011001
01001100101
11100010101
10010001101
10100100011
01010010011
11001001011
00111000111
00000111111
11111111111

(j is unknown.)

Fig. 2.3 Continued

a modification is possible only when the investigator can control the computer by hand during the computation in the sence of man-machine cooperation. M(10,5) and M(11,5) were found by such a method.

Note that $C_1(7,3)$ is the famous Hamming code and $C_1(n,d)$ is the group code in general. (See § 2.3.3).

§ 2.3.3 Minimum Distance Group Codes

A minimum distance code with specified minimum distance d is called the minimum distance group code if it is the group and denoted by $G(n,d)$.

In the case of the group code the minimum distance d is connected to the error correcting ability by the following relation. A code is the t-error correcting code if and only if t is the largest weight of coset leaders such that all error patterns of errors $\leq t$ are coset leaders, or the minimum distance is $2t+1$ or $2t+2$.

Maximum minimum distance group codes (M.M.D.G.C.)

The maximum dinimum distance group code is a $G(n,d)$ for which $[G(n,d)]$ is the largest among all $G(n,d)$s. The search for M.M.D.G.C. for arbitrarily given n and d has been made by many authors but not solved generally. That is, the general algorithm was not established to calculate the order of M.M.D.G.C. for every n and d but only some entries were found by applying various methods proposed by many authors. It is said that the general algorithm would not exist. We expected that S.P. be the general one but it was found to be also a partial solution of the problem.

The following theorem, which was found by the present author and proved by S.Kitawaki, gives an interesting relationship between the structure of the group and that of the order as to the metric in the space of X^n.

THEOREM: The code $C_1(n,d)$ which is generated first by S.P. is the group code.

This theorem is proved by the mathematical induction of n. For $m < n$, $C(m,d)$ is isomorphic to $C(n,d)$ if the specified n-m columns of $C(n,d)$ are all zero and the rests are identical with $C(m,d)$. In the proof such codes are identified. $C_1(n,d)$ is denoted by $C(n,d)$ for the sake of brevity.

PROOF

(1) $C(d,d)$ is clearly the group.

(2) Suppose that $C(m,d)$ is the group and

$$C(m,d) = \left\{ a_0, a_1, \ldots, a_{2^k-1} \right\}.$$

Let a' be the first element which is generated by S.P. after $C(m,d)$ was generated. That is,

$$a' = \min \left\{ a: d(a, C(m,d)) \geq d, \ a \in X^{m+r} \right\}, \qquad (P1)$$

where $r \geq 1$ and r is the smallest number for which a' exists. Then it is sufficient to show that $C(m+1,d)$ is the group. Or

$$C(m+1,d) = \left\{ a_0, a_1, \ldots, a_{2^k-1}, \right. \qquad (P2)$$
$$\left. a'+a_0, a'+a_1, \ldots, a'+a_{2^k-1} \right\}$$

In other words we should indicate that by S.P. the elements $a'+a_0$, $a'+a_1$, .., $a'+a_{2^k-1}$ are generated in this

order after $C(m,d)$ was generated. This is proved in the following step.

(i) It is clear that $a' = a'+a_0$ is generated first after a_{2^k-1} from (P1).

(ii) Let a'_i be the i-th element which is generated by S.P. after a_{2^k-1} and suppose that $a'_i = a'+a_i$ for $i = 0, 1, 2, \ldots, j$, where $a'_0 = a'$.

(iii) Now (P3) is to be proved.

$$a'_{j+1} = a'+a_{j+1} \tag{P3}$$

To prove (P3), it is noted at first that

$$d(a'+a_{j+1}, \{ a_0,\ldots, a_{2^k-1}, a',\ldots, a'_j \}) \geq d,$$

since $d(a', \{ a_0,\ldots, a_{2^k-1} \}) \geq d$ or $\| a'+a_i \| \geq d$ for $i = 0,1,..,2^k-1$. Therefore

$$a'+a_{j+1} \geq a'_{j+1}. \tag{P4}$$

Furthermore, since $d(a'_{j+1}, a_i) \geq d$ for $i = 0,\ldots,j$,

$$a'_{j+1}+a' \geq a'_{j+1}. \tag{P5}$$

Finally

$$a_{j+1}+a'_{j+1} \geq a'. \tag{P6}$$

In fact, $C(m,d) = C(m,d)+a_{j+1}$

and

$$d(C(m,d), a'_{j+1}+a_{j+1}) = d(C(m,d)+a_{j+1}, a'_{j+1}+a_{j+1})$$

$$= \min \| a+a'_{j+1} \| \text{ for } a \in C(m,d) \geq d.$$

From the following lemma it is seen that three inequalities (P4), (P5) and (P6) hold simultaneously if and only if all equalities hold for them. In particular,

$$a'+a_{j+1} = a'_{j+1},$$

what was to be proved.

LEMMA: The following three inequalities (P7),(P8) and (P9) hold simultaneously, if and only if all equalities hold for them.

$$x + y \geq z \qquad \text{(P7)}$$

$$z + x \geq y \qquad \text{(P8)}$$

$$y + z \geq x \qquad \text{(P9)}$$

where x, y and z are elements of X^n.

PROOF OF LEMMA

If one equality, say $x+y = z$, holds, the other two also hold. In fact from $(x+y)_k = z_k$, it is clear that $(z+x)_k = y_k$ and $(y+z)_k = x_k$. Now it is shown that the inequalities

$$x + y > z \qquad \text{(P10)}$$

$$z + x > y \qquad \text{(P11)}$$

$$y + z > x \qquad \text{(P12)}$$

do not hold simultaneously.

Suppose for example $x+y > z$. Then there is an integer k such that $(x+y)_i = z_i$ for $i = k+1, k+2, \ldots, n$ and $(x+y)_k = 1$, $z_k = 0$. Therefore $(z+x)_i = y_i$ and

$(y+z)_i = x_i$ for $i = k+1, k+2, \ldots, n$. Since $(x+y)_k = 1$, either $x_k = 1$ and $y_k = 0$ or vise versa. If $x_k = 1$ and $y_k = 0$, then $(y+z)_k = 0 < 1 = x_k$, which contradicts (P12). If $x_k = 0$ and $y_k = 1$, then $(x+z)_k = 0 < 1 = y_k$, which contradicts (P11). This completes the proof of the lemma.

Algorithm generating $C_1(n,d)$ effectively[44]

Some computations indicated that $C_1(n,d)$ seems to be M.M.D.G.C.. Though this postulate was found incorrect, $C_1(n,d)$ is still the interesting code. So a more effective algorithm generating $C_1(n,d)$ only was required. The algorithm is based on the theorem just proved.

Though $G(n,d)$ is the code in the n-dimensional linear space, it is considered to be a code in the m ($m < n$)-dimensional space, if all components of some n-m columns are zero. According to this convention, in the following equations will appear some terms whose dimensions are apparently different. The same convention will be used in the notation of the code word.

$$\text{(i)} \quad G(d,d) = \begin{matrix} 000\ldots000 \\ \underbrace{111\ldots111}_{d} \end{matrix} \tag{2.9}$$

Clearly $[G(d,d)] = 2$.

$$\text{(ii)} \quad G(i+1,d) = G\big(G(i,d),\ g_k\big), \qquad \text{for } i = d, d+1, \ldots, n-1 \tag{2.10}$$

where $G\big(G,\ g\big)$ means the group which is generated by

the generators of G and g, and g_k is the k-th generator, which satisfies the following condition:

$$g_k = \min \{ a: \ d(a, G(i,d)) \geq d, \ a \in X^{i+1} \}. \quad (2.11)$$

The distance between an element and a subset of X^n is defined as usual. If there is no element which satisfies the condition (2.11), let $G(i+1,d) = G(i,d)$ and try to find g_k newly. It is seen that $[G(i+1,d)] = [G(i,d)] + 1$, if g_k is found in X^{i+1} and $[G(i+1,d)] = [G(i,d)]$ otherwise. This completes the algorithm.

Results of computation

It is easy to program the algorithm for the computer. For the group codes we used IBM-7090, which has about 35,000 36-bit words. The program is written in the symbolic coding system FAP. The results are listen in Table 2.2, which shows $C_1(n,d)$s. Fig. 2.4 illustrates the algorithm using the example of $G(n,7)$. As seen from the definition of the algorithm, codes with shorter length are obtained at the intermediate stages of generation of longer ones. Fig. 2.5 shows the generators of $C_1(19,5)$, $C_1(23,7)$ and $C_1(22,9)$. It took about two hours for 7090 to generate them. A great portion of the computation time was spent for finding the new generator. Investigating Table 2.2 $C_1(18,5)$ was found not to be the MMDGC for $n = 18$ and $d = 5$. For this case there is the better code found by Griesmer, which has 2^{10} elements.

The maximality of $C_1(12,5)$, $C_1(19,5)$ and $C_1(21,9)$ is doubtful. The other codes are seen to be maximum. The posturate that $C_1(n,d)$ is MMDGC for any n and d is not correct, but S.P. generates relatively many MMDGCs by a single principle.

Example: G(14,7)

G(7,7) = 0000000 0000000
0000000 1111111

G(8,7) = G(9,7) = G(10,7) = G(7,7)

g_2 = 000 11110000111

G(11,7) = 000 00000000000
000 00001111111
000 11110000111
000 11111111000

G(12,7) = G(11,7)

g_3 = 0 1100110011001

G(13,7) = 0 0000000000000
0 0000001111111
0 0011110000111
0 0011111111000
0 1100110011001
0 1100111100110
0 1111000011110
0 1111001100001

g_4 = 10101010101010

G(14,7) = 00000000000000
00000001111111
00011110000111
00011111111000
01100110011001
01100111100110
01111000011110
01111001100001
10101010101010
10101011010101
10110100101101
10110101010010
11001100110011
11001101001100
11010010110100
11010011001011

[G(14,7)]= 2^4 = 16

Fig. 2.4 Illustration of algorithm

n =	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
d = 5	2	2	2	4	4	8	16	16	32	2^6	2^7	2^8	2^9	$\underline{2^9}$	2^{10}				
d = 7			2	2	2	2	4	4	8	16	32	32	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}
d = 9					2	2	2	2	2	4	4	4	8	8	16	32	32	2^6	
d = 11							2	2	2	2	2	2	4	4	4	8	8	16	

$[C_1(n,d)]$ obtained by computation

Table 2.2

g_1 000000000000011111
g_2 000000000011100011
g_3 000000001100100101
g_4 000000010101001010
g_5 000001100000100110
g_6 000010100001001011
g_7 000100100100101011
g_8 000100010010100110 1
g_9 001000010010110011 1
g_{10} 110000000000100111

d = 5

g_1 000000000000000011111111
g_2 000000000000111100001111
g_3 000000000110011001100 1
g_4 000000000101010101010 10
g_5 000000001001011001101 00
g_6 000000110000001101010 11
g_7 000001010000010101100 01
g_8 000010010000011000111 01
g_9 000100010001000101111 00
g_{10} 001000010001001000011 10
g_{11} 010000010001010001001 11
g_{12} 100000010001011100100 10

d = 7

g_1 000000000000011111111 1
g_2 000000001111100000111 1
g_3 000001110001100011001 1
g_4 000110010010100101010 1
g_5 001010100101001010101 0
g_6 110000010011000110101 0

d = 9

Fig. 2.5 Generators of $C_1(n,d)$

CHAPTER III

A Trial and Error Model for Approximation of Boolean Functions

§ 3.1 Introduction

The motivation of this study is to formulate an aspect of the pattern recognition. In the study of the pattern recognition the Boolean function plays an important role. In the ordinary methodology the investigator observes the object patterns and establishes the recognition principle in the form of the logical decision function. If the recognition process could be represented completely by the Boolean functions which represent the characteristic features, the recognition machine could be constructed using the logical circuit.

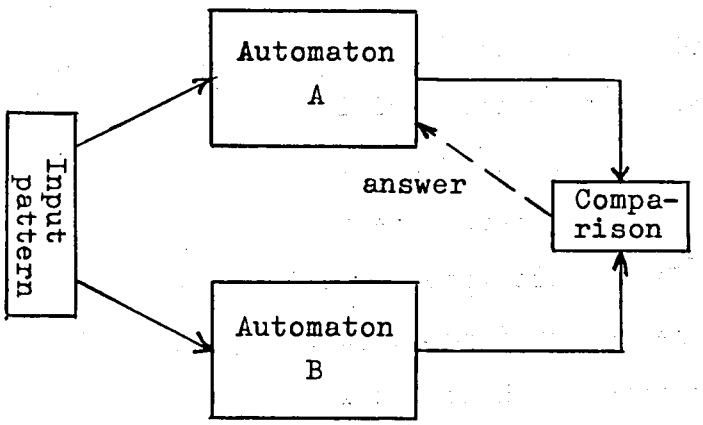
In determining the decision scheme the average rate of the correct decision is wanted to be as high as possible. This is because the pattern is of statistical nature. The investigator investigates many pattern samples before determining the decision criterion. And the recognition machine itself has not the mechanism to observe samples and to determine the criterion, once the machine was constructed.

Contrary to this we do try to formulate the process of the establishment of the decision criterion itself. In our foumulation the pattern is identified with a

Boolean function and it is the objective of the study to get a model for approximating an arbitrary function by means of the trial and error procedure.

The trial and error model

The feature of our model is that it consists of the probabilistic logical elements. We do not claim the superiority of the probabilistic element over the deterministic one in the field of so called "learning". But it could be expected that the model would serve the statistical decision theory. Note that the introduction of the probabilistic element was voluntary, contrary to the study of the reliability, where the element is inevitably regarded as probabilistic.



Trial and error model

Fig. 3.1

Fig. 3.1 shows the block diagram of our model. The automaton A is the machine which are going to be designed for the trial and error model. The automaton B is the other automaton, which has a decision criterion for the pattern recognition, even if it can not be expressed explicitely in the form of the logical function. It is supposed that B gives the answer, "Good" or "Bad" only, to the automaton A, according to its decision if the pattern is the considered pattern or not. It is our intention to design A so that it may distinguish the considered pattern from many patterns using only the answers of B. In this case it is not required that A itself be the good discriminator but it is sufficient to construct A so that it may be just the designer of the discriminator. (See § 3.6.)

Like the ordinary automaton the model is composed of many elements, each of which has the same structure. The basic element is the probabilistic (stochastic) logic element.

Correspondence between the code and the logical function

Before entering the theme, the correspondence between the code and the logical function is noted, because the code theory is expected to be useful in the field of the pattern recognition.

The logical function, more precisely the Boolean function, is represented in many different forms accord-

ing to applications. In this study it is represented using the terms of the code theory. A function $F(x)$ is called the n-variable Boolean function, if x is an element of the space X^n and F takes 0 or 1 as its function value. The function is denoted by $F(x_1, x_2, \ldots, x_n)$, where X_i is the variable. F is defined completely by the function values for all n-tuples of X^n. This may be done by giving the set of the n-tuples for which F takes the value 1. Therefore a Boolean function of n variables can be identified by a block code of length n. (See § 2.2).

The function space of all n-variable Boolean functions which contains 2^{2^n} functions, is denoted by Ω_n and any subset of Ω_n can be considered as a code of length 2^n. In this correspondence it should be noted that the order of the components of each n-tuple is significant for representing the Boolean function, contrary to the case of the code word, since the equivalence of code words does not hold for the function in general.

§ 3.2 Pattern and Boolean function

What is called the pattern in general could not be expressed by a single mathematical formula but would be more complicated object. In order to have insight to the study, however, we should restrict our interest to the simplified model, for instance such a pattern as represented by logical functions. In this study we treat

the (combinatorial) Boolean function as a tool for representing the pattern. Therefore the practical patterns, such as the written character, the spoken speech sound, the photograph of the bubble chamber and so on, can not be applied to the model as they are.

A reason for using the Boolean function is that it is closely connected to the digital technique which is very powerful in constructing the machine practically. The recognition machines constructed so far by many investigators are basically related to the logical circuit, in particular the decision circuit is composed of logical elements.

In this section the pattern is defined and the process of the pattern recognition formulated as that of the approximation of the Boolean function.

Pattern

Let n points be fixed in a certain space. To each point the random variable x_i, which takes only the value 1 or 0, is attached. The observation is to see if $x_i = 1$ or 0 at the given instance. The n-tuple $(x_1, x_2, \ldots, x_n)$ is called the image where x_i is the sample value of each point. A fixed set of images is called a pattern. That is, to a pattern belong many images in general. The pattern discrimination is then to decide to which pattern the observed image belongs.

The pattern can be identified with the n-variable

Boolean function $F(x_1, x_2, \ldots, x_n)$, which is defined as follows: If a n-tuple $(x_1^0, x_2^0, \ldots, x_n^0)$ is the image belonging to the pattern, then $F(x_1^0, x_2^0, \ldots, x_n^0) = 1$, and otherwise, $F(x_1^0, x_2^0, \ldots, x_n^0) = 0$. In order that F is the well defined function, it is necessary that every n-tuple is said to belong to the pattern or not.

As seen from definition it would be possible that two distinct patterns in the ordinary sense have the same functional representation. The choice of the space where the observed points lie and that of the places of the observed points are decisive for the whole recognition system. The most common configuration of the observed points is the two dimensional mesh with photoelectric elements for the recognition of the printed characters. This problem should be solved case by case and can not be treated generally. Our study starts, therefore, with the assumption that the observed points have been fixed by some other consideration.

Let the information source be $(A, p(a))$, where A is the finite set of the objects which are to be recognized and $p(a)$ is the probability that the object a appears. Let $p(x/a)$ be the conditional probability that $x (= x_1, x_2, \ldots, x_n)$ is observed when a was generated from the source. $p(x) = \sum_{a \in A} p(a)p(x/a)$ is the probability that the image x is observed anyhow.

If $p(x/a) = 1$ and $p(x/\ ) = 0$ for every other object in A, the observation of x indicates uniquely the appearance of a. In general, this is not the case because of the limited number of the observed points and existence of the noise in the observing machine.

We agree that the difference of two patterns with respect to the observed points is defined by that of two Boolean functions.

Let Ω_n be the set of all n-variable Boolean functions. Then the metric d is defined in Ω_n by the formula (3.1).

$$d(F, G) = \sum_{x \in X^n} |F(x) - G(x)| \, p(x), \qquad (3.1)$$

where $p(x)$ is the probability distribution defined on X^n. It is easily seen that d actually satisfies the three axioms of the metric with the exception that if $d(F,G) = 0$, then $F = G$ almost surely.

The appearance of images at the observed points is regarded as the stochastic process. The process is assumed to be stationary and independent as to the time.

The problem can be regarded as to extract the Boolean function, which might represent the original pattern best, from infinitely many observed images utilizing the answers of the reference automaton B. Then it is formulated in terms of the metric d as follows: to find the Boolean function F_0 which is nearest to the considered pattern. If the function F_0 is found, a

logical circuit corresponding to it is constructed. This inputs to this circuit is the n-tuple (x_1, x_2,..., x_n) of the observed points. Then it can be expected that F_0 gives the best guess as to the considered pattern. The procedure to get F_0 for any pattern is as follows:

(I) For any positive number e and any function F_0, to find the function F such that $d(F_0, F) \leq e$.

This could not be solved in general because the observed points are not necessarily enough. Therefore the first compromise is made.

(II) For any F_0, to find the function F_i out of the given k functions F_1, F_2, ..., F_k such that $d(F_0,F_i)$ is the smallest. The set of the given functions is denoted by Φ.

The present study is an answer to this phase of problem.

(III) To find the function F which is nearer to F_0 than F_i of problem (II).

In problems (II) and (III) it is desirable that the approximation is uniform as to the considered pattern and the probability distribution p(x). In order to solve these problems it is necessary to investigate the set Φ in the space Ω_n. For this we employed the theory of group code as shown in CHAPTER IV.

§ 3.3 The stochastic logical circuit

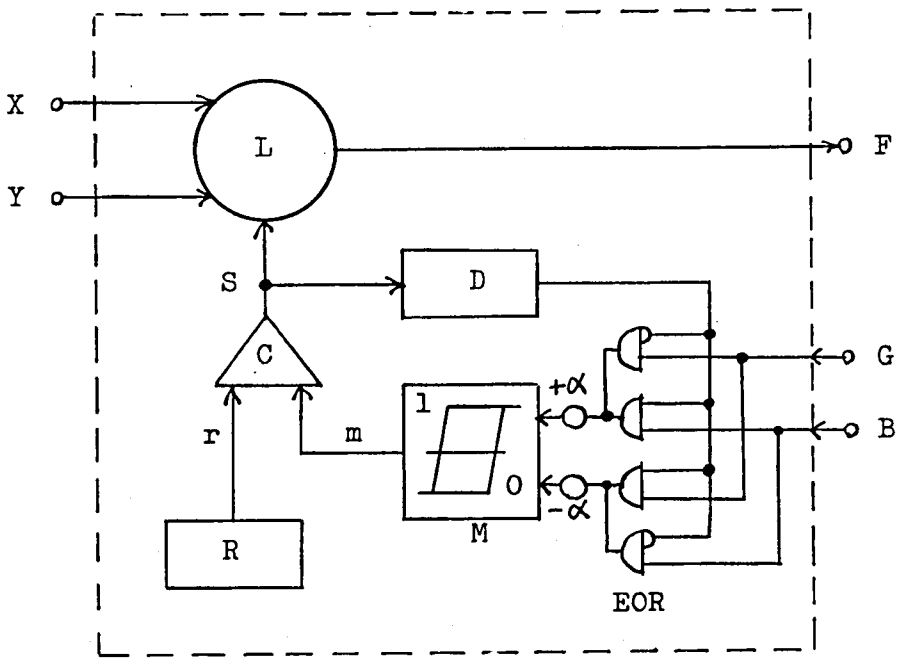
The trial and error model should have some active mechanism to do the "trial". For this purpose we devised a probabilistic element described below. The element takes one of two 2-variable Boolean functions with the probability which changes according to the history of the element. By combining several such elements we get a logical net which takes one of some functions with the probability determined by its history.

§ 3.3.1 The stochastic logic element (S.L.E.)

The stochastic logic element is a logic element in the sense that the inputs to it are logical variables and its output is also a logical variable. Contrary to the ordinary (deterministic) element, however, its function is not the single Boolean function but composed of two functions. The S.L.E. takes one of two 2-variable Boolean functions with probability p and the other with probability $1-p$, where p changes according to the history of the element.

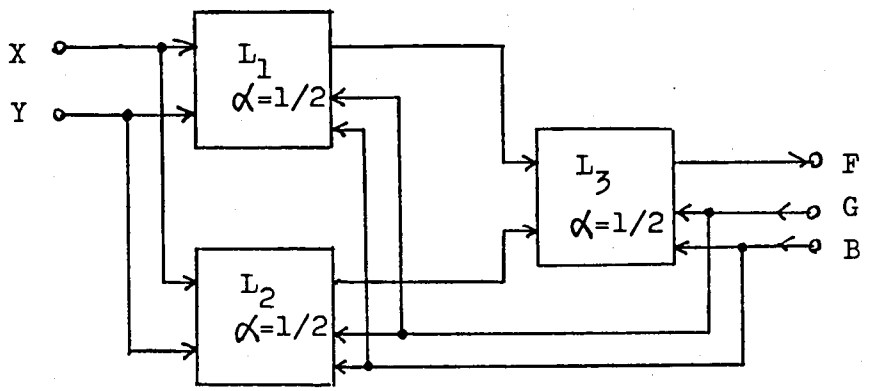
Fig. 3.2 is the block diagram of the S.L.E. Referring to it, X and Y are the inputs and F is the output. G and B are the terminals where the "answer" is given. Each part is explained roughly below:

L: The logical element which takes one of two Boolean functions $f_0(X,Y)$ and $f_1(X,Y)$ according that the value of S is 0 or 1 respectively, f_0 and f_1



L: logic element D: delay element
C: comparator R: random number generator
M: memory element EOR: exclusive OR circuit

Fig. 3.2 Stochastic logic element



L_1: $f_0=X\bar{Y}$, $f_1=X+\bar{Y}$ L_2: $f_0=XY$, $f_1=X+Y$

L_3: $f_0=\bar{X}Y$, $f_1=\bar{X}+Y$

Fig. 3.3 Model of stochastic logic net

may be arbitrary but fixed for a S.L.E.

R: The random number generator which generates an independent random number r, which is distributed between 0 and 1 uniformly, at each trial.

M: The memory with saturation characteristics whose content m is between 0 and 1. m is increased or decreased by an amount α according to the result of the trial. If m reaches the value 1 then m remains 1, even if another α is added. But if α is subtracted, then it is reduced at once by that value. ($0 < \alpha < 1$).

C: The comparator which gives 1 as its output S, if $r \geq m$ and 0 otherwise.

D: The delay element which holds the value S during the period between the trial and the answer.

EOR: The logical circuit which generates the value α or $-\alpha$ according to the combination of the answer and the output of D.

We used the computer to simulate the behavior of S.L.E. instead to construct it by means of the real physical element. When making S.L.E. by the electronic circuit, there are difficulties in devising the memory element M.

Behavior of S.L.E.

At a certain instance the inputs are given to the terminals X and Y. At the same time a random number

r is generated by R and r is compared with the content m of M. Then the output S of C is 1, if $r \geq m$ and is 0 if $r < m$. L takes one of the functions f_0 and f_1 according to S = 0 or S = 1. The function value for the selected function and the inputs appears at F. During this process the time delay is considered to be negligible. This is one phase of the trial.

After a time interval the answer is given the terminal G or B. Then the input to M is determined by EOR from the answer and the output of D. If G = 1, the content m of M changes so that the same S as that of the previous trial may be expected at the next trial much and if B = 1, m changes conversely. If there is no answer or are both answers at the terminals, M remains unchanged. When m has changed, one cycle of the behavior ends.

The inner state of S.L.E. is completely specified by m. The number of states depends on the initial value of m and α. If α is small, m takes many values. After several trials it is expected that m reaches one of both saturation limits. If m does not divide 1, there are much states. In our first model α is assumed to be 1/2. Therefore after some trials the possible values of m are 0, 1/2 and 1, whatever the initial value is. Thus if α were 1/3, they would be 0, 1/3,

2/3 and 1.

§ 3.3.2 The stochastic logic net (S.L.N.) (in case of two variables)

The stochastic logic net is a logical net which consists of S.L.E. and ordinary logic elements as well as the ordinary delay element. The main feature of the connection is that S.L.N. has the circuit for the answer. Fig. 3.3 shows an example of S.L.N. using three S.L.E.s, where the lines connecting G-B terminals are for the answer. As is easily imagined, the behavior of the S.L.N. is so complicated that it may not be analized generally.

We use the model shown in Fig. 3.3 to indicate that S.L.N. is useful for the trial and error process. The model is the case for two variable functions. For every element $\alpha = 1/2$. Each element has different functions for L as indicated at the bottom of the Figure.

In the following sections the process is analyzed for the model shown in Fig. 3.3.

§ 3.3.3 The behavior of S.L.N.

The state of the circuit of Fig. 3.3 is determined by those of memory elements. It is denoted by the vector notation $\bar{m} = (m_1, m_2, m_3)$, where m_i is the state of the i-th element. In the case of $\alpha = 1/2$, there are 27 states, among which only 9 states are realized in the trial and error process as shown later. The set of ran-

dom numbers generated by Gs is denoted by $\bar{r} = (r_1, r_2 r_3)$, where r_i is the random number generated by the i-th element. The set of outputs of the comparators is represented by $\bar{S} = (S_1, S_2, S_3)$, where S_i is the output of the i-th comparator. There are 8 possible values for $\bar{S}$. The function of the circuit is represented by $\bar{L}$ which varies according to $\bar{S}$. If for example, $\bar{S}$ =(1, 1,0), $\bar{L}$ is the function of f_1 of the first element, f_1 of the second element and f_0 of the third element.

The connection of the answer is made so that the G terminal of the circuit goes to every G terminals and the B terminal to every B terminals.

The behavior of this model is explained as follows: First the inputs are given at a certain instance, then the set of random numbers $\bar{r}$ is generated and the vector $\bar{S}$ is determined. The function $\bar{L}$ is determined by it and the output of this function appears at F for the given inputs. This is the phase of the trial. Then after a while the answer is given G or B. Then the state $\bar{m}$ changes according to it. This completes one cycle of the trial and answer. In order to use the model as the trial and error process, the answer is given by Automaton B as follows: If $F_0(x,y) = 1$, where F_0 is the decision criterion of B in the form of the Boolean function, then the answer G is given. Otherwise the answer B is given. In the real process the criterion may be unknown in many cases.

§ 3.4 Analysis by the Markov chain[56~61]

In order to analyse the stochastic behavior of the circuit we employ the "trial-output table" as indicated in Table 3.1. The trial-output table is determined, when the circuit is fixed, in the following way: In the first row all possible combinations of S are listed. The second row is the function $\bar{L}$ which corresponds to the $\bar{S}$ over it. The i-th column is the function F_i which is extended according to inputs. Thus the i-row j-column element e_{ij} is the function value of F_j for the i-th input. For example, if $XY = 10$, and the trial $\bar{S} = 011$, then the output of the circuit is $e_{27} = 1$.

$\bar{S}$	000	100	010	110	001	101	011	111
XY \ $\bar{L}$	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8
00	0	0	0	0	1	0	1	0
10	0	0	0	0	0	0	1	1
01	0	0	1	1	1	1	1	1
11	1	0	1	0	1	1	1	1

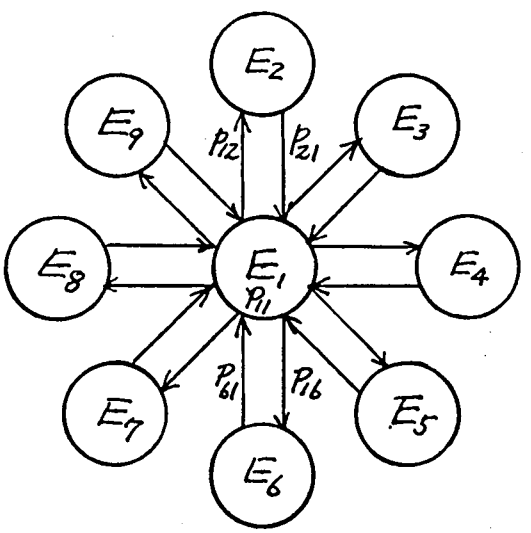
Table 3.1

The trial-output table for Fig. 3.3

Since the state of the circuit is stochastically dependent only on the immediately previous state, the tran-

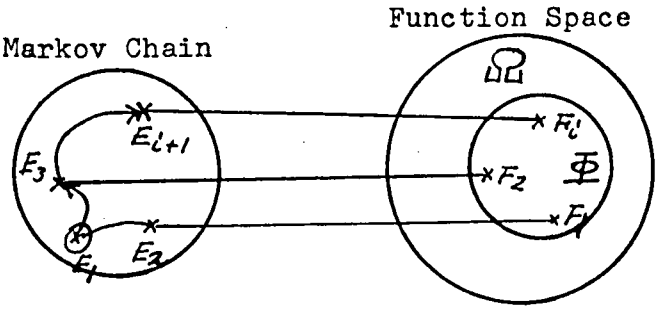
sition of the state can be represented by the simple Markov chain. We are interested in the transition of the state and not in the output itself. For the present model the number of the states which the circuit takes becomes 9 after a few trials, because $\alpha = 1/2$ and Ms are the saturating type memories. They are (1/2, 1/2, 1/2), (1,1,1), (0,1,1), (1,0,1), (0,0,1), (1,1,0), (0,1, 0), (1,0,0), (0,0,0). They are denoted by E_1, E_2, ..., E_9 respectively. The state transition diagram of the Markov chain is indicated in Fig. 3.4. The transition occurs only to the direction of the arrow as seen from the fact that M is the saturation type memory and $\alpha = 1/2$.

At the states other than E_1, all Ms are saturated and the result of the next trial is predicated completely.



State diagram for the net of Fig. 3.3

Fig. 3.4



Correspondence between Markov chain and function space

Fig. 3.5

Therefore the state can be regarded as corresponding to the function. In fact E_{i+1} corresponds to F_i uniquely. E_1 does not correspond, however, to any function.

Suppose now the function F_0 is fixed as the decision criterion of Automaton B and kept constant during the trial and error experiment. The Markov chain is stationary if the input is stationary and F_0 is constant but it is nonstationary if F_0 changes, even when the input is stationary. (See §3.7). Denote $\{ F_i \}$ by Φ.

The transition probability is calculated from the input probability p and F_0 using the trial-output table. Let F_0 be represented by the vector (f_1, f_2, f_3, f_4), where $f_1 = F_0(0,0)$, $f_2 = F_0(1,0)$, $f_3 = F_0(0,1)$ and $f_4 = F_0(1,1)$. Let the input probability distribution be represented by (p_1, p_2, p_3, p_4), where $p_1 = p(0,0)$, $p_2 = p(1,0)$, $p_3 = p(0,1)$ and $p_4 = p(1,1)$,

and $\sum_{i=1}^{4} p_i = 1$, $p_i \geq 0$. Using e_{ij} ($i = 1, 2, 3, 4$ and $j = 1, 2, \ldots, 8$) we defined another entity c_{ij} by

$$c_{ij} = 1 \quad \text{if} \quad e_{ij} = f_i,$$
$$c_{ij} = 0 \quad \text{if} \quad e_{ij} \neq f_i. \tag{3.2}$$

Then the probability p_{ij} that the state E_i goes to E_j is given by the following formulas.

$$p_{11} = 0,$$
$$p_{1(j+1)} = 1/8 \sum_{k=1}^{4} \left\{ c_{kj} + (1-c_{k(9-j)}) \right\} p_k, \quad \text{for } j = 1,2, \ldots, 8,$$
$$p_{jj} = \sum_{k=1}^{4} c_{k(j-1)} p_k \quad \text{for } j = 2,3, \ldots, 9,$$
$$p_{j1} = 1-p_{jj} \quad \text{for } j = 2,3, \ldots, 9. \tag{3.3}$$

For example, $F_0 = XY$ and $p_1 = p_2 = p_3 = p_4 = 1/4$, then the stochastic matrix is calculated as shown in Fig. 3.6A. In this chain the state E_2 is the absorbing state, because $p_{22} = 1$. Note that $F_0 = F_1$. This case is called the model 1. The model 2 is $F_0 = \bar{X} + \bar{Y}$, for the same uniform input probability distribution, whose matrix is Fig. 3.6B. The model 3 is indicated in Fig. 3.6C, for which $F_0 = X \cdot \bar{Y} + \bar{X} \cdot Y$ and $p_1 = 0$, $p_2 = p_3 = p_4 = 1/3$. These two models are the ergodic chains, since F_0 does not equal any F_i.

$$P = \begin{pmatrix} 0 & 3/16 & 3/16 & 1/8 & 1/8 & 1/8 & 1/8 & 1/16 & 1/16 \\ 0 & 1 & & & & & & & \\ 1/4 & & 3/4 & & & & & & \\ 1/4 & & & 3/4 & & & & & \\ 1/2 & & & & 1/2 & & & & \\ 1/2 & & & & & 1/2 & & & \\ 1/4 & & & & & & 3/4 & & \\ 3/4 & & & & & & & 1/4 & \\ 1/2 & & & & & & & & 1/2 \end{pmatrix}$$

Stochastic matrix of model 1

Fig. 3.6A

$$P = \begin{pmatrix} 0 & 1/16 & 1/16 & 1/8 & 1/8 & 1/8 & 1/8 & 3/16 & 3/16 \\ 1 & 0 & & & & & & & \\ 3/4 & & 1/4 & & & & & & \\ 3/4 & & & 1/4 & & & & & \\ 1/2 & & & & 1/2 & & & & \\ 1/2 & & & & & 1/2 & & & \\ 3/4 & & & & & & 1/4 & & \\ 1/4 & & & & & & & 3/4 & \\ 1/2 & & & & & & & & 1/2 \end{pmatrix}$$

Stochastic matrix of model 2

Fig. 3.6B

$$P = \begin{bmatrix} 0 & 1/24 & 1/12 & 1/8 & 1/6 & 1/12 & 1/8 & 1/6 & 5/24 \\ 1 & 0 & & & & & & & \\ 2/3 & & 1/3 & & & & & & \\ 2/3 & & & 1/3 & & & & & \\ 1/3 & & & & 2/3 & & & & \\ 2/3 & & & & & 1/3 & & & \\ 2/3 & & & & & & 1/3 & & \\ 1/3 & & & & & & & 2/3 & \\ 1/3 & & & & & & & & 2/3 \end{bmatrix}$$

Stochastic matrix of model 3

Fig. 3.6C

Since the repeated trials are characteristic to the trial and error process, it is necessary to analyze the behavior of the model after many trials. The limiting theory of the Markov chain is useful in this case.

In general the trial and error model is analysed by its stochastic matrix. Using Feller's terminology[56] the chains appearing in our models are classified into two classes, the reducible chain and the irreducible chain. If there is a function in Φ which is almost equal to the considered pattern F_0, then the chain is absorbing. There may be more than one absorbing states according to the structure of the S.L.N. In the case of the irreducible chain, all states are persistent non-null. Furthermore if the period is 1, then the chain is called

ergodic. It seems that in our model the periodic chain does not appear. Therefore we analyse the absorbing and the ergodic chains exclusively.

In the study of the trial and error model the stationary distribution, $\lim_n P^n$, the speed of the convergence of it and P^n have significance. As seen later we can measure the stationary distribution approximately. It is almost clear that the speed of the convergence is essential to the model. In the following the analysis is shown using the models Fig. 3.6A, B and C.

The methods are:

(1) The stationary distribution q for the ergodic chain is calculated by solving the set of simultaneous linear equations.

(2) P^n is calculated by means of the Sylvester's expansion theorem of the matrix.

(3) The speed of the convergence is estimated by the absolute values of eigenvalues of the said simultaneous equations and by the Sylvester's theorem.

The stationary distribution $q = (q_1, q_2, \cdots, q_9)$ is shown in Table 3.2A, B and C as well as the distance distribution $d_i = d(F_0, F_i)$. Note that to the state E_1 no function corresponds.

P^n is calculated by the direct matrix multiplication when n is small but it is not practical for large n. Feller indicated the computation method for the case

至	—	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8
d	—	0	1/4	1/4	1/2	1/2	1/4	3/4	1/2
$\bar{q}$	0	1	0	0	0	0	0	0	0

Stationary distribution and distance distribution of model 1. F_0 = (0001).

Table 3.2A

至	—	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8
d	—	1	3/4	3/4	1/2	1/2	3/4	1/4	1/2
$\bar{q}$	.322	.020	.027	.054	.081	.081	.054	.242	.121

Stationary distribution and distance distribution of model 2. F_0 = (1110).

Table 3.2B

至	—	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8
d	—	1	2/3	2/3	1/3	2/3	2/3	1/3	1/3
$\bar{q}$	.304	.013	.038	.057	.152	.038	.057	.152	.190

Stationary distribution and distance distribution of model 3. F_0 = (0110).

Table 3.2C

that the eigenvalues of the matrix P are all simple roots. In our models the multiple roots appear, so another method is required. For this we employed the Sylvester's theorem[60], which allows the expansion of the matrix by means of its eigenvalues.

Let $\alpha_1, \alpha_2, \ldots, \alpha_i$ be the eigenvalues of P and $s_1, s_2, \ldots, s_i$ their multiplicities. If P has full degeneracy for all eigenvalues, i.e. $\text{rank}(P - \alpha_r I) = a - s_r$ for any r, where a is the degree of P, then

$$P^n = \sum_{r=1}^{i} \alpha_r^n [K(\alpha_r)]^{(s_r)}, \qquad (3.4)$$

where

$$[K(\alpha_r)]^{(s_r)} = \sum_{k \neq r}^{i} \frac{(\alpha_k I - P)^{s_k}}{(\alpha_k - \alpha_r)^{s_k}}$$

The simple root has full degeneracy. Since P is the stochastic matrix, for every eigenvalue α_r, $|\alpha_r| \leq 1$, and among α_rs there is the eigenvalue which equals 1. This may be denoted by α_1. If there are k absorbing states then $s_1 = k$, but for this $\alpha_1 (= 1)$ the full degeneracy holds. For the ergodic chain α_1 seems to have the full degeneracy. If the full degeneracy does not hold, the equation (3.4) becomes complicated so that the convergence of $\lim P^n$ becomes slow. In discussing $\lim P^n$ the full degeneracy needs to be considered only for the root α_r such that $|\alpha_r| = 1$.

According to the theorem $\lim P^n$ is represented as follows.

$$\lim_{n\to\infty} P^n = \sum_{r=s_1+1}^{i} \frac{(\alpha_r I - P)^{s_r}}{(\alpha_r - 1)^{s_r}},$$

provided that $\alpha_r \neq 1$ for $r \neq 1$ and the full degeneracy holds for α_1.

The speed of the convergence is estimated by the inequality (3.5).

$$\left|(P^n - \lim_{n\to\infty} P^n)_{kl}\right| \leq \sum_{r=2}^{i} |\alpha_r|^n (K_r)_{kl}$$

$$k,l = 1,2, \ldots,i \quad (3.5)$$

where K_r is the matrix whose elements are the absolute values of those of $[K(\alpha_r)]^{(s_r)}$ and kl means the k-1 element of the matrix.

That is the speed has the order of the exponential function of the eigenvalues. If there is a root whose absolute value is large, the speed is slow. For example, the convergence of P^n is indicated in Fig. 3.7 for the model 1 and the model 3. The characteristic equations for them are:

<u>model 1</u>

$$(P - \alpha I) = (1-\alpha)(1/2-\alpha)^2(3/4-\alpha)^2(0.287-\alpha)$$
$$\cdot(0.358+\alpha)(0.618-\alpha)(0.952-\alpha)$$

model 3

$$(P - \alpha I) = (1-\alpha)(1/3-\alpha)^3(2/3-\alpha)^2(0.033-\alpha) \cdot(0.510-\alpha)(0.543+\alpha).$$

The greatest root for the model 1 is 0.952 and that of model 3 is 0.667. Therefore the model 1 converges slower than the model 3, as seen from Fig. 3.7 also.

§ 3.5 Approximate functions

In this section the relationship between the stationary probability distribution and the distance distribution of Φ is investigated. The stationary distribution q is regarded as that of Φ, instead that of the states $E_1, E_2, \ldots, E_9$.

Then the following relation holds from Tables 3.2A, B and C.

Let $d_i = d(F_0, F_i)$ and q_i be the stationary distribution of E_i.

$$\text{If } d_i \leq d_j \text{ then } q_{i+1} \geq q_{j+1} \quad (i,j = 1,2,\ldots,8) \tag{3.6}$$

where the equalities do not necessarily hold simultaneously. In particular for the function which has the smallest distance corresponds to the greatest probability. We assured from other several models that the relation (3.6) holds in general.

Therefore the problem II of § 3.2 is solved by know-

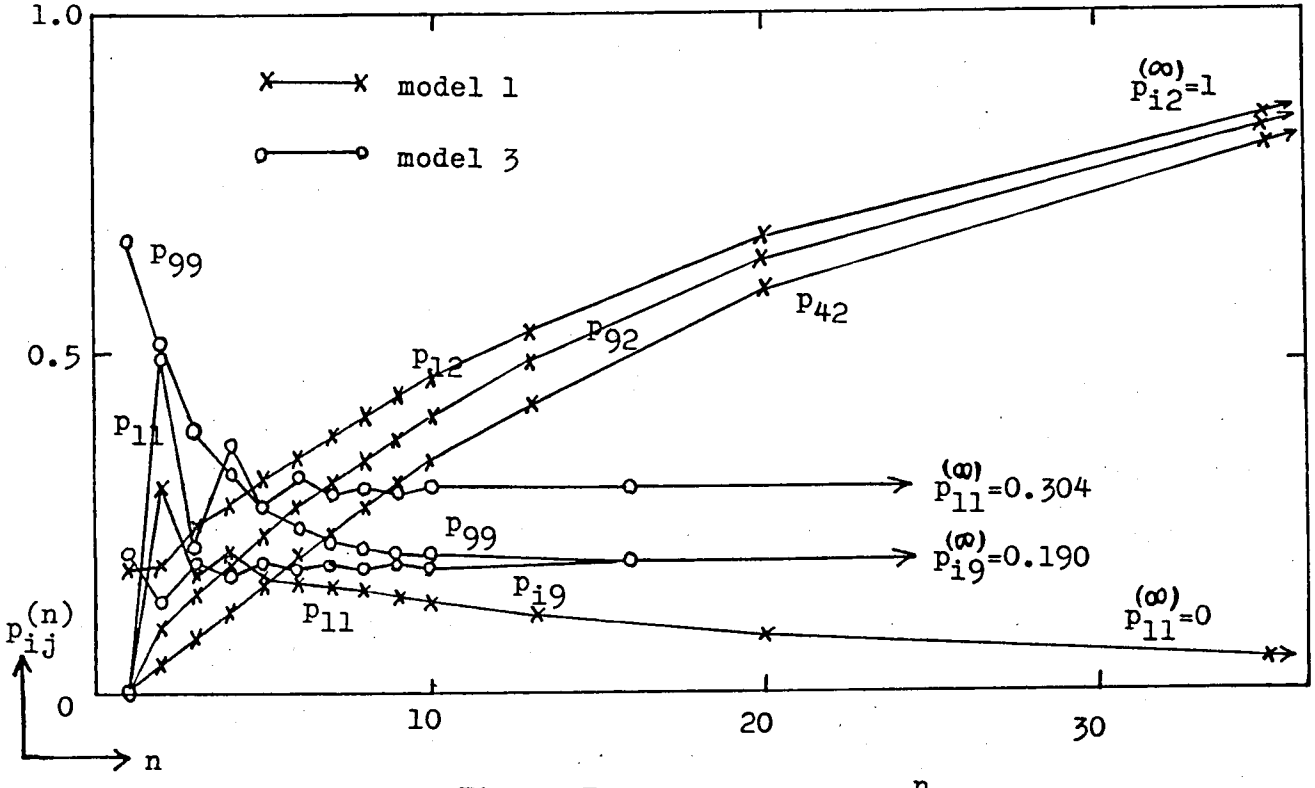


Fig. 3.7 Convergence of P^n

ing the stationary distribution. For our models 1, 2 and 3, the approximate functions are F_1, F_7 and F_8, and the differences are 0, 1/4 and 1/3 respectively.

In order that the relation (3.6) holds generally, it is desirable that the set of functions $\mathfrak{F}$, which the circuit has, distributes uniformly in the space Ω_n. In this respect we established a method to find the epsilon net with specified properties, using the theory of group codes. The next chapter is devoted to this theme.

However the complete systematic design of the set $\mathfrak{F}$ and that of the S.L.N. which realizes it have not been reached.

§ 3.6 Measurement of the stationary distribution[57,58]

In the preceding section the model of the trial and error was analyzed, provided that the pattern F_0 is known and fixed. Here we should treat the practical problems arising in the experiment. That is, the pattern is unknown and to be recognized by the model.

As is indicated in § 3.5 it is required to measure the stationary probability distribution of the Markov chain. For this purpose we measure the present state of the circuit $\bar{m}$. Then by calculating the relative frequency of each state we obtain the distribution approximately. What guarantees this method is the strong law of large numbers (ergodic theorem) of the Markov chain.

Assume that the chain is ergodic and let $S_i(n)$ be the number of times that the circuit takes the state E_i during n trials, then

$$P\left\{ \lim_{n\to\infty} \frac{S_i(n)}{n} = q_i \right\} = 1. \qquad (3.7)$$

This means that almost all experiments converge to the stationary distribution $\bar{q}$. If we do not distinguish the absorbing states each other, this method is applicable to the absorbing chain. In the case of the periodic chain, however, another measurement is necessary to know the period.

At the practical measurement the speed of the convergence of the ergodic theorem (3.7) is essential, because we want to obtain the stationary distribution as fast as possible. In other words the chain which converges quickly is desirable. But there has been not an effective estimation of the speed in general. We found an estimation for the case of the absorbing chain[47].

Suppose the chain is absorbing with the absorbing state E_i and the condition of the full degeneracy is satisfied for the eigenvalues α_r of the stochastic matrix of the chain (See § 3.5). Let the initial state of the chain be E_k. Then the following inequality holds for the limit probability q_i (=1).

$$P\left\{\left|\frac{S_n(i)}{n} - q_i\right| < e, \quad \text{for any} \quad n > m\right\} \leq p_{ki}^{(em)}$$

$$\leq 1 - \sum_{r=2} |\alpha_r|^{em} k_r \qquad (3.8)$$

where $p_{ki}^{(em)}$ is the transition probability from E_k to E_i after em trials and k_r is the constant of order 1. Note again that $|\alpha_r| < 1$.

This inequality gives an effective estimation for our models.

We constructed the simulation system of the trial and error model by the digital computer and experimented the process to see that it really converges. The system consists of the part representing the S.L.E. and the part indicating the connection of the elements. The compiler produces the simulating program from the given connection.

In the case of the experiment using hardwares, the measurement of $S_i(n)$ would not be easy. To use the counter is not desirable from the economical point of view.

Now we indicate the fact that it is not preferable to use the S.L.N. for descriminating the patterns. The model is designed only for the extraction of the function and the practical pattern descrimination should be done by the deterministic logical circuit, which is design by means of the model.

Let the input probability distribution be $p = (p_1, p_2, p_3, p_4)$ for the model of Fig. 3.3 and the stationary distribution be $\bar{q} = (q_1, q_2, \ldots, q_9)$. Then the probability P_G that the output of the S.L.N. coincides with the decision by F_0 is calculated using the trial-output table.

$$P_G = \sum_{i=1}^{8} (q_{i+1} + q_1/8) \cdot (\sum_{k=1}^{4} c_{ki} \cdot p_k) \qquad (3.9)$$

On the other hand,

$$d(F_0, F_i) = 1 - \sum_{k=1}^{4} c_{ki}\, p_k \qquad (3.10)$$

Therefore

$$P_G \leq \max_i (\sum_{k=1}^{4} c_{ki}\, p_k) \left\{ \sum_{i=1}^{8} (q_{i+1} + q_1/8) \right\}$$

$$= \max_i (\sum_{k=1}^{4} c_{ki}\, p_k) = 1 - \min_i d(F_0, F_i) \qquad (3.11)$$

Therefore if F_0' is the approximate function, then

$$P_G \leq 1 - d(F_0, F_0'). \qquad (3.12)$$

This proves the assertion.

§ 3.7 The non-stationary Markov chain

Thus far the criterion function F_0 has been assumed to be constant during the trial and error experiment even though it is unknown. In the practical pattern recogni-

tion, however, the criterion of the Automaton B could not be assumed to be constant (from the point of view that it is a logical function). Furthermore if the Automaton B is a man, it may be possible that he replies the Automaton A by mistake. This situation is formulated as the non-stationary Markov chain. Here the input probability is assumed to be stationary.

In general the non-stationary process has not been investigated fully. So the study of each case is necessary by its own method. We need only to see that the process does not converge or converges slowly, if such situation occurs. Especially the limit distribution is disturbed by the incorrect answer.

A numerical experiment is shown below in order to explain some phenomena arising in the case of the non-stationary chain.

Let F_0 and F_0^* be two different criterion functions. Suppose the Automaton B applies F_0 for a pattern, i.e. $F_0(x) = 1$ if the input x belongs to the pattern. Besides this, it uses (by mistake) F_0^*, but not so often. In this case we consider F_0^* is the noise of this process. In this case, if F_0^* is not applied after finite number of trials, then the process will be the same as the undisturbed trial and error process, since the limit distribution of the ergodic and the absorbing chains is independent from the initial state. Contrary to this, if F_0^*

appears for ever, then the limit distribution of the chain changes and the trial and error process is affected. In this case the resolution (differences of q_i) becomes less and the extraction of the proper approximate function becomes difficult. Consider, for example, the extreme situation that the Automaton B changes his criterion definitely from F_0 to F_0^* or his object pattern changes from F_0 to F_0^*. Then the Automaton A should indicate also some change of its behavior, and give the approximate function of F_0^* as the result.

Now let P and P^* be the stochastic matrix of the models for F_0 and F_0^* respectively. The process above described is represented by the multiplication of matrices. At the step where F_0^* is applied the matrix P^* is multiplied in place of P. Therefore, for example, PPPPPP* PPPPPP*PPP... shows that at the sixth and the twelfth trials the Automaton B made mistake.

Fig. 3.8 shows the convergence of the chain with P^* multiplied periodically every fifth times and tenth times. The model is that which is shown in Fig. 3.3 and F_0 = XY = 0001 and F_0^* = Y = 0011. The limit distribution of the process can be obtained by considering the chains $P' = P^4P^*$ and $P' = P^9P^*$ for the every fifth time noise model and the every tenth time noise model respectively. The limit distribution does not exist in the strict sense. From the figure it is seen that the model which was absorbing becomes the non-absorbing model. (See the curve

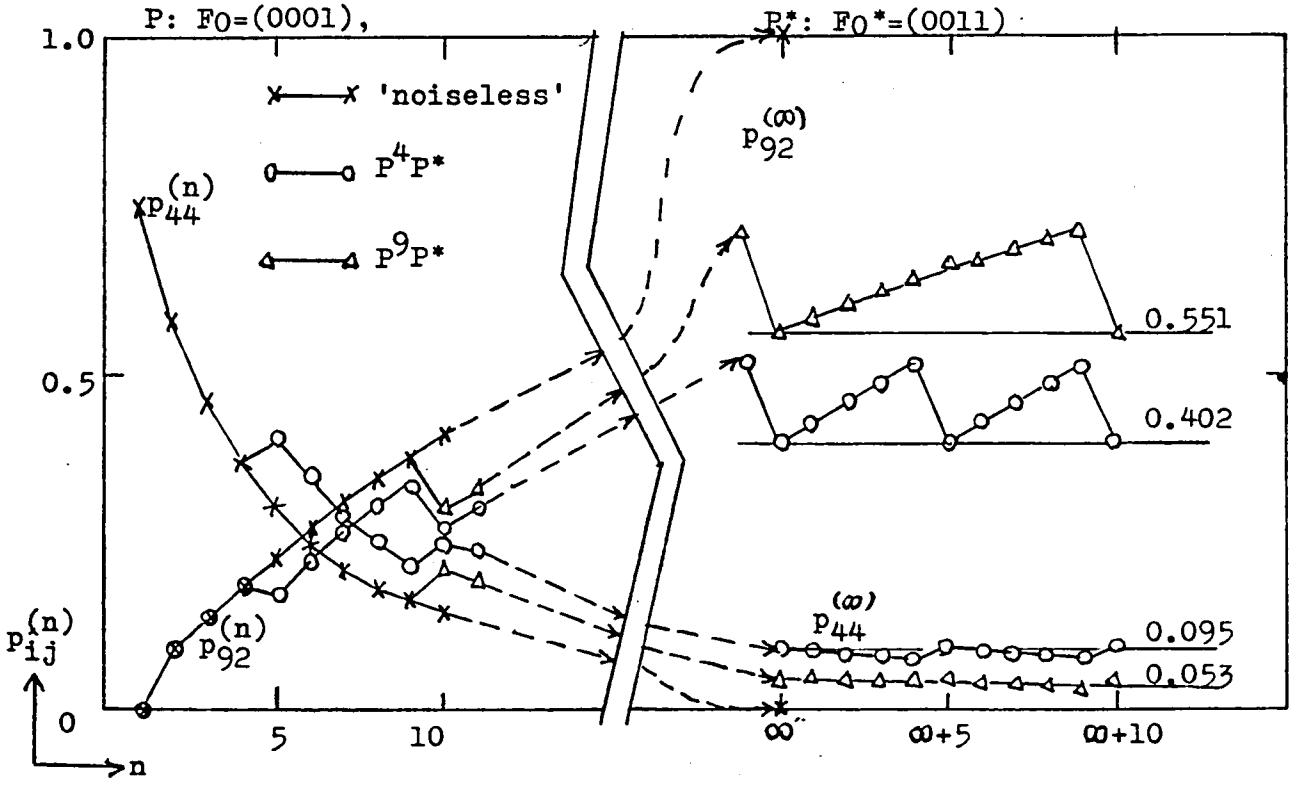


Fig. 3.8 Convergence of non-stationary Markov chain

of p_{92}). The oftener the noise occurs, the less the differences of q_is are. Therefore the noise makes the behavior disordered. Thus the non-stationary Markov chain is the model of the real trial and error process. So the study of it is necessary, in particular we should assure experimentally that the ergodic theorem holds approximately, because it is the basis of the measurement as described in § 3.6.

CHAPTER IV

An Application of the Group Code Theory to the Approximation of Boolean Functions

§ 4.1 Introduction[50, 51]

As is mentioned in §3. it is necessary to obtain the set of functions Φ which will give the uniform approximation in the space of the n-variable Boolean functions Ω_n. The epsilon net which is used in the theory of approximation is an appropriate tool for this purpose. It was found, however, difficult to solve the approximation problem generally, so the epsilon group net was defined and used together with the fruitful theory of group codes.

§ 4.2 Definitions[55]

The distance d is defined by (4.1).

$$d(F(x), G(x)) = \sum_{x \in X^n} |F(x)-G(x)| \cdot \mu(x) \qquad (4.1)$$

where F and $G \in \Omega_n$ and $\mu(x)$ is the probability distribution on X. x is the element of the space $X = X_1 \times X_2 \times , \ldots, \times X_n$ where $X_i = \{0, 1\}$.

Let Φ be a subset of Ω_n. Then to each Φ corresponds a real number ε between 0 and 1 such that:

$$\varepsilon = \operatorname*{Max}_{F \in \Omega_n} \operatorname*{Min}_{G \in \Phi} d(F, G) \qquad (4.2)$$

Then we have three definitions concerning ε .

Definition 1 For a real number ε if there exists a subset Φ of Ω_n such that (4.2) holds, then we call Φ a ε-net (epsilon net).

Definition 2 For a given ε ($0 \leq \varepsilon \leq 1$), a ε-net is called the smallest ε-net, if it contains the least number of elements. Denote this number by N_ε .

Definition 3 For a given integer N ($1 \leq N \leq 2^{2^n}$), a subset Φ of N elements is called the best net (BN) of size N, if its corresponding ε is the smallest among all the subsets of Ω_n containing N elements. Denote this smallest number by ε_N.

The definitions depend on the space Ω_n, i.e. n and $p(x)$, but we do not refer to it when there is no danger of confusion. As is easily seen, N_ε is essentially the same as $N_\varepsilon^W(A)$ defined by Kolmogoroff et al.[55], though we have defined it independently and in a different form. In general there could be many smallest ε-nets and best nets of size N for a given ε and a given N respectively.

In general, it has been found that it is rather difficult to get an effective method to obtain ε_N and N_ε for arbitrarily given ε and N even when $\mu(x)$ is the uniform distribution. Such an algorithm that one examines all possibilities one after another takes so much time and memory of the machine that it is hardly practical.

This may be the same difficulty as that encountered in the study of the error correcting code. Therefore we restrict our concern to a certain kind of net and apply the group code theory to it.

We found it convenient to introduce the structure of the group into the space Ω_n in the same way as we did with the group code in §2.2 . As usual the relationship between the metric and the group is defined by means of a norm as follows:

$$d(F(x), G(x)) = \| F(x)-G(x) \| \qquad (4.3)$$

where $\| \ \|$ means the norm. It seems natural to define the group of modulo two sum because the Boolean function is defined on the product space of the Boolean ring $\{0,1\}$. A some what formal reasoning to do so is given below.

A uniqueness of the modulo two sum

Let (X, d) be the metric space with finite underlying set X. Denote $X= \{ x_1, x_2, \ldots, x_n \}$ and $d(x_i, x_j) = d_{ij}$. Then the following two propositions are proved.

P1 In order that a group operation $x_i \cdot x_j$ and a real number (norm) $\| x_i \|$ can be defined for elements of X such that they satisfy the relation

$$d_{ij} = \| x_i \cdot x_j^{-1} \| , \qquad (4.4)$$

the following condition is necessary. The n-tuple of numbers ($d_{i1}, d_{i2}, \ldots, d_{in}$) is a permutation of (d_{11},

$d_{12}, \ldots, d_{1n}$) for all $i = 1, 2, \ldots, n$.

P2 (the case of discrete metric)

If $d_{1j} \neq d_{1k}$ for any different j and k, the metric is called discrete and at most one group operation can be defined and satisfies the relation (4.4) stated in P1. If there is such an operation, then it is commutative and $x \cdot x = 1$ (the identity element) for any x. In this case the norm $\| x \|$ is also determined uniquely.

PROOF: Suppose $x_1 = 1$. Then

$$\| x_i \| = \| x_i \cdot x_1^{-1} \| = d_{il} = d_{1i} = \| x_1 \cdot x_i^{-1} \| = \| x_i^{-1} \|.$$

From the condition $d_{1i} \neq d_{1j}$, if $\| x_i \| = \| x_j \|$, then $x_i = x_j$. Therefore if $d_{ij} = d_{1k}$, then $x_i \cdot x_j^{-1} = x_k$ uniquely.

In particular, $x_1 \cdot x^{-1} = x$ and $x \cdot x_1^{-1} = x$. And from the symmetry of d, $x_i \cdot x_j^{-1} = x_j \cdot x_i^{-1}$. This completes the proof.

Now in our case of "weighted Hamming distance", the metric satisfies the condition of P1 of course. If (x) is not a special distribution, then the metric is discrete. Therefore the modulo two sum is the unique group operation in sum form. If $\mu(x)$ is uniform, i.e. d is the ordinary Hamming distance, however, other group operations besides modulo two sum can be defined which also satisfy the relation (4.4) in P1. Considering the above results it seems natural to introduce the group of modulo

two sum.

From here on we assume Ω_n is such a group with coefficients GF(2) and the norm $\| F \|$ is defined on Ω_n so that (4.3) holds. Then we have

Definition 4 Replacing "subset Φ " by "subgroup Φ " in the Definitions 1 to 3, we obtain definitions for " ε-group net (ε-GN)", "the smallest ε-GN" and "the best GN (BGN) of size N" respectively. N must be 2^k.

Two reasons to have investigated the group net are that even the study of the group net gives insight to the more general case and that the group is useful in applications. Now it would be instructive to give some examples of the group net: the set of all linear functions (more generally, the set of all functions of degree $\leq i$) and the set of all functions of i variables, where $i \leq n$, in the space Ω_n. We will discuss these examples later.

§ 4.3 Application of the group code theory[8]

In this section several theorems are proved using the decoding scheme described in § 2.2.

Theorem 1:

Let Φ be a subgroup of order 2^k in Ω_n. Partition Ω_n into cosets relative to Φ . In each coset we choose an element which has the smallest "norm" among the elements in the same coset and call it the coset leader.

Let $a_0, a_1, \ldots, a_{m-1}$ ($m = 2^{2^n-k}$) be the coset leaders. Then Φ is a ε-GN of Ω_n, where

$$\varepsilon = \max \| a_i \| , \qquad 0 \le i \le m-1.$$

Proof

Let $\Phi = \{ \varphi_0, \varphi_1, \ldots, \varphi_\ell \}$. If $x \in \Omega_n$ belongs to the i-th coset $a_i \cdot \Phi$, then

$$d(x, \Phi) = \min_k d(x, \varphi_k) = \min_k \| x \cdot \varphi_k^{-1} \| = \min \| x \| = \| a_i \|$$

Therefore $\varepsilon = \max_x d(x, \Phi) = \max_i \| a_i \|$.

This completes the proof.

In the followings every time we write "partition into cosets", it means the partition stated in Theorem 1. This theorem is useful in obtaining the epsilon by a computer when a subgroup is given.

In order to apply the results of the group code theory and to get powerful theorems, we must have some restrictions on the probability distribution $\mu(x)$ of X^n. A probability distribution is called "partially uniform", if there exists a subset A of X^n such that $x \in A$, $\mu(x) = 0$; $x \notin A$, $\mu(x) = 1/2^n - M$, where M is the number of elements in A. If A is empty, then $\mu(x)$ is uniform.

If $\mu(x)$ is partially uniform, Ω_n can be regarded as the set of all vectors whose components are 0 or 1

and whose length is 2^n-M. The norm correponds to the weight. Any subset of Ω_n corresponds to a code. Therefore a GN is equal to a group code of length $\bar{n} = 2^n$-M, where the metric is equivalent to the "Hamming distance".

From here on the metric space Ω_n is assumed to be the same as a code and the norm is changed according to the metric. The group operation is, however, unaltered.

Theorem 2:

If $\mu(x)$ is partially uniform and subgroup Φ equals a perfect or quasi-perfect code whose order is 2^k, then Φ is the BGN of size 2^k. Furthermore, for a given ε, if there is a ε-GN which is equal to a perfect code, then it is the smallest ε-GN.

Proof

In the definition of the perfect or quasi-perfect code let t be the number such that all vectors of norm less than or equal to t are coset leaders and some of the vectors of norm t+1 are not. Then the epsilon for this code is t+1 (or t) from Theorem 1. No other group of the same order gives a smaller epsilon. Therefore Φ is the best group net of its size. The latter half of the theorem for the case of the perfect code is obvious from the meaning of the word "perfect".

Note that the latter half does not hold necessarily in the case of the quasi-perfect code. A counter example is given by Example 4 below.

$\bar{n}$	i	$\binom{\bar{n}}{i}$	k=2	3	4	5	6	7	8	9	10
4	0	1	1	1							
	1	4	3								
5	0	1	1	1							
	1	5	5	3							
	2	10	2								
6	0	1	1	1	1						
	1	6	6	6	3						
	2	15	9	1							
7	0	1	1	1	1	1					
	1	7	7	7	7	3					
	2	21	18	8							
	3	35	6								
8	0	1	1	1	1	1	1				
	1	8	8	8	8	7	3				
	2	28	28	20	7						
	3	56	27	3							
9	0	1	1	1	1	1	1	1			
	1	9	9	9	9	9	7	3			
	2	36	36	33	22	6					
	3	84	64	21							
	4	126	18								
10	0	1	1	1	1	1	1	1	1		
	1	10	10	10	10	10	10	7	3		
	2	45	45	45	39	21	5				
	3	120	110	64	14						
	4	210	90	8							
11	0	1	1	1	1		1	1	1	1	
	1	11	11	11	11		11	11	7	3	
	2	55	55	55	55		20	4			
	3	165	165	126	61						
	4	330	226	63							
	5	462	54								
12	0	1	1	1				1	1	1	1
	1	12	12	12				12	12	7	3
	2	66	66	66				19	3		
	3	220	220	200							
	4	495	425	233							
	5	792	300								

Coset leaders of optimal codes [8]

Table 4.1

Table 4.1 is a copy of Slepian's table[8] showing the coset leaders of the optimal codes for the symmetric binary channel. From it, using Theorem 2, we can find the BGN for smaller n and k: For some larger n and k, the table of the quasi-perfect code given by W.W.Peterson[54] is useful.

Now some examples are given concerning Theorem 2.

Example 1

Suppose $n = 3$ and $M = 1$, therefore $\bar{n} = 7$. Let Φ be the Hamming code of order 16. Then Φ is the 1-GN of Ω_3. That is, any function of Ω_3 is within the distance 1 from Φ. This is clear from Theorem 1 and from the fact that the Hamming code is the 1-error correcting perfect code. Furthermore Φ is the BGN of size 16 and conversely the smallest 1-GN is the Hamming code of order 16.

Example 2

Suppose $n = 3$ and $M = 0$, therefore $\bar{n} = 8$ (the uniform distribution). Let's take a subgroup Φ of order 8. According to Table 4.1, the norms of the coset leaders of the optimal code $C(8,3)$ are 0, 1, 2 and 3. Thus $\max \| a_i \|$ equals to 3. This code is not quasi-perfect. But it is a 3-GN and the BGN of the size 8. (See Theorem 4). It is also seen from the same table that the smallest 3-GN is not of order 8 but of order 4.

Example 3

Suppose $\bar{n} = 8$. The quasi-perfect code of order 16 is the BGN and $\mathcal{E} = 2$. Furthermore this is the smallest 2-GN of size 16.

Example 4

Suppose $\bar{n} = 9$. The quasi-perfect code of order 32 is the BGN and $\mathcal{E} = 2$ but it is not the smallest 2-GN. The smallest 2-GN is the code of order 16.

The following two simple theorems are also useful in obtaining the BGN. We assume here also the uniform distribution.

Theorem 3

Let the order of Φ be N, where $N = 2^k$. If Φ is a $\mathcal{E}$-GN, then

$$\mathcal{E} \geq t+1$$

for such a t that

$$\sum_{i=0}^{t+1} {}_{\bar{n}}C_i \geq 2^{\bar{n}-k} \geq \sum_{i=0}^{t} {}_{\bar{n}}C_i$$

Theorem 4

If Φ is the optimal code and there exists a number t such that all vectors of norm $\leq t$ and some of those of norm t+1 and t+2 are coset leaders of the partition of Ω_n into cosets relative to Φ and none of the vectors of norm $\geq t+3$ are the coset leaders, then Φ is the BGN of its size. Note that the optimal code is not necessarily the BGN.

n \ k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0															
2		0														
3			0													
4	2	1	1	0												
5	2	2	1	1	0											
6	3	2	2	1	1	0										
7	3	③	2	1	1	1	0									
8	4	3	3	2	1	1	1	0								
9	4	4	③	2	2	1	1	1	0							
10	5	④	4	③	2	2	1			0						
11	5	5	④	3		2					0					
12	6	⑤	④									0				
13													0			
14						3		2						0		
15									2						0	
16																0
17									3							
18																
19																
20																
21												3		2		
22												3			2	
23												4				2
24												4				
25												4				
26																

The $\mathcal{E}_N$ ($N=2^k$) of the BGN for the partially uniform distribution

Table 4.2

These theorems are easily proved from the distribution of norms of the coset leaders. For example in theorem 4, $\mathcal{E} = t+2$ from Theorem 1 and if there were a code with $\mathcal{E} = t+1$ then it must be a quasi-perfect code (therefore optimal), so we would result in contradiction.

Using Theorems 1 to 4 Table 4.2 showing the BGNs for the partially uniform distribution was obtained. In obtaining BGNs for larger $\bar{n}$ and k we used the figures given by Peterson[54]. In Table 4.2 the blank means that the BGN has not been obtained because the corresponding quasi-perfect code is unknown for this parameter. The encircled entry was obtained by Theorem 3 and the underlined one by Theorem 4 and the others by Theorem 2.

Using Table 4.1 we can also compute the order of the smallest $\mathcal{E}$-GN. Table 4.3 shows the logarithms to the base 2 of the orders of the smallest $\mathcal{E}$-GNs. Table 4.4 gives the same figure as Table 4.3 for the normalized $\mathcal{E}$. From Table 4.5 we know for example that the order of the smallest $\mathcal{E}$-GN for $1/4 \leq \mathcal{E} \leq 3/8$ in Ω_3 is 2^4. It is also seen that for all $\bar{n}$ the order of the smallest $\mathcal{E}$-GN for $\mathcal{E}$ larger than 1/2 is 2. Kolmogoroff et al. defined " $\mathcal{E}$-entropy" of a metric space. Since we restricted the net to the group net, Table 4.5 shows the entity which is similar to the $\mathcal{E}$-entropy but larger than it.

$\bar{n}$ \ ε	1	2	3	4	5	6
4	2	1				
5	3	1	1			
6	4	2	1			
7	4	3	1			
8	5	4	2	1		
9	6	4	3	1		
10	7	5	4	2	1	
11	8	5/6	4	3	1	
12	9	7		3	2	1

$k = \log_2 N_\varepsilon$ of the smallest ε-GN.

Table 4.3

$\bar{n}$	N_ε: for $\leq \varepsilon \leq$
4	4: $0 \leq \varepsilon < 1/4$, 2: 1/4 1/2, 1: 1/2 1
5	5: 0 1/5, 3: 1/5 2/5, 1: 2/5 1
6	6: 0 1/6, 4: 1/6 1/3, 2: 1/3 1/2 1: 1/2 1
7	7: 0 1/7, 4: 1/7 2/7, 3: 2/7 3/7 1: 3/7 1
8	8: 0 1/8, 5: 1/8 1/4, 4: 1/4 3/8 2: 3/8 1/2, 1: 1/2 1
9	9: 0 1/9, 6: 1/9 2/9, 4: 2/9 1/3 3: 1/3 4/9, 1: 4/9 1
10	10: 0 1/10, 7: 1/10 1/5, 5: 1/5 3/10 4: 3/10 2/5, 2: 2/5 1/2, 1: 1/2 1
11	11: 0 1/11, 8: 1/11 2/11, ? ? 4: 3/11 4/11, 3: 4/11 5/11, 1: 5/11 1
12	12: 0 1/12, 9: 1/12 1/6, ? ? 3: 1/3 5/12, 2: 5/12 1/2, 1: 1/2 1

The $N_\varepsilon (=2^k)$ for the smallest ε-GN relative to the normalized distance.

Table 4.4

§ 4.4 Application of the $\mathcal{E}$-GN

It is noted at first that the results obtained in §4.3 are independent of the permutation of the code components. That is, we can choose an arbitrary representation of a function by means of a code word.

We agree to represent a function as follows:

$$F(X,Y,\ldots,Z) = (F(00\ldots0), F(100\ldots0), F(010\ldots0), \ldots, F(111\ldots1)) \qquad (4.4)$$

If $\mu(x) = 0$ for some x, then we delete the corresponding code component. According to this notation, the system of all linear functions in Ω_3, for example, is represented by Fig. 4.1.

0	00000000	1	11111111
X	01010101	1+X	10101010
Y	00110011	1+Y	11001100
Z	00001111	1+Z	11110000
X+Y	01100110	1+X+Y	10011001
Y+Z	00111100	1+Y+Z	11000011
X+Z	01011010	1+X+Z	10100101
X+Y+Z	01101001	1+X+Y+Z	10010110

Fig. 4.1

The set of all functions of two variables X and Y in the space Ω_3 is generated by the generators shown in Fig. 4.2.

0	00000000
X	01010101
Y	00110011
X.Y	00010001
1	11111111

Fig. 4.2

Now suppose Φ is the set of all linear Boolean functions in n variables and $\mu(x)$ is the uniform distribution. Since Φ is a subgroup of Ω_n, we can find the epsilon ε' for Φ according to Theorem 1. From Table 4.2, on the other hand, we know the epsilon ε of the BGN for $\bar{n} = 2^n$ and $k = n+1$. Comparing ε' and ε we can see if the set of linear functions is the BGN of the whole space or not.

For an example consider the case $n = 3$. The order of the group of linear functions is $2^4 = 16$. From Table 4.2 we see $\varepsilon = 2$ for this case, i.e. we can get at most the approximation of 1/4 by means of linear functions. It is seen that the set of linear functions is the 2-GN and therefore one of the BGNs of size 16 in Ω_3. If we add one bit to the Hamming code so that the resultant code has the even parity, then one representation of this code is the set of linear functions, i.e. the set generated by X, Y, Z and 1.

The set of all functions of degree ≤ 2 has the order 2^7. From Table 4.2 the BGN has $\varepsilon = 1$. On the other hand the stated set is a 1-GN of Ω_3, therefore is the BGN.

Though we have not obtained the BGN for larger value of $\bar{n}$, it seems that the validity of these propositions, i.e. "the linear functions give the BGN" and so on, is doubtful.

In this study we obtained the interesting results only for the space with the partially uniform distribution. The group code theory could be applied to such a limited case.

Though the study of the epsilon entropy seems to be interesting and useful in the information theory, we are far from the general solusion. The problem to obtain N in general seems to be almost impossible to solve.

The application of the epsilon group net was rather unsystematic and limited.

LITERATURE

PART I

[1] R.W.Hamming (1950) Error detecting and error correcting codes: B.S.T.J. 29 147-160.

[2] E.N.Gilbert (1952) A comparison of signaling alphabets: B.S.T.J. 31 504-522.

[3] - - - (1958) Gray codes and paths on the n-cube: B.S.T.J. 31 815-825.

[4] M.J.E.Golay (1949) Notes on digital coding: Proc. IRE. 37 657- .

[5] - - - (1954) Binary coding: PGIT-4 23-28.

[6] I.S.Reed (1954) A Class of multiple-error-correcting codes and the decoding scheme: PGIT-4 38-49.

[7] P.Elias (1954) Error-free coding: PGIT-4 29-37.

[8] D.Slepian (1956) A Class of binary signaling alphabets: B.S.T.J. 35 203-234.

[9] - - - (1956) A note on two binary signaling alphabets: PGIT-2 84-86.

[10] R.R.Varshamov (1957) Estimate of the number of signals in error-correcting codes: D.A.N.SSSR. 117 no.5 739-741.

[11] S.P.Lloyd (1957) Binary block coding: B.S.T.J. 36 517- .

[12] W.Ulrich (1957) Non-binary error correction codes: B.S.T.J. 36 1341-1388.

[13] C.Y.Lee (1958) Some properties of nonbinary error-correcting codes; PGIT-4 77- .

[14] N.Wax (1959) On upper bounds for error detecting and error correcting codes of finite length: PGIT-5 168-174.

[15] A.B.Fountaine and W.W.Peterson (1959) Group code equivalence and optimum codes: PGIT-5 60-70.

[16] H.S.Shapiro and D.L.Slotnick (1959) On the mathematical theory of error-correcting codes: IBM J. 3 25-34.

[17] E.McClusky (1959) Error correcting code-a linear programming approach: B.S.T.J. 38 1485-1512.

[18] M.Plotkin (1960) Binary codes with specified minimum distance: PGIT-6 445-450.

[19] C.Campopiano (1960) Construction of relatively maximal systematic codes of specified minimum distance from linear recurring sequences of maximal period: PGIT-6 523-528.

[20] D.D.Joshi (1960) A note on upper bounds for minimum distance codes: Information and Control 1 289-295.

[21] J.H.Griesmer (1960) A bound for error correcting codes: IBM J. 4 532-542.

[22] J.E.MacDonald (1960) Design methods for maximum minimum-distance-error correcting codes: IBM J. 4 43-57.

[23] R.P.Bambah, D.D.Joshi and S.Luthar (1961) Some lower bounds on the number of code points I: Information and Control 4 313-319.

[24] - - - - - - - - - (1961) Some lower bounds on the number of code points II: Information and Control 4 320-323.

[25] J.Lipp (1960) Upper bounds for error detecting and correcting codes: PGIT-6 correspondence 557-8.

[26] R.G.Fryer (1960) Note on "On upper bounds for error detecting and error correcting codes of finite length: PGIT-6 correspondence 502

[27] P.T.Strait (1961) A lower bound for error-detecting and error-correcting codes: PGIT-7 114-115.

[28] W.W.Peterson (1962) Bounds for error-correcting codes: PGIT-8 correspondence 60.

[29] N.Lawrence (1962) Upper bounds for error-detecting and error-correcting codes: PGIT-8 correspondence 58.

[30] M.Nadler (1962) A 32-point n=12, d=5 code: PGIT-8 correspondence 58.

[31] L.D.Grey (1962) Some bounds for error-detecting codes; PGIT-8 200-202.

[32] S.M.Johnson (1962) A new upper bound for error-correcting codes: PGIT-8 203-207.

[33] - - - (1963) Improved asymptotic bounds for error-correcting codes: PGIT-9 198-205.

[34] J.E.Bartow (1963) A reduced upper bound on the error correction ability of codes: PGIT-9 correspondence 46.

[35] E.Myravaagnes (1963) On maximum-weight codes: PGIT-9 correspondence 289-290.

[36] Y.Komamiya (1964) General theory of most efficient codes: Report 163 Digital Comp. Lab. Univ. Ill.

[37] C.V.Freiman (1964) Upper bounds for fixed-weight codes of specified minimum distance: PGIT-10 correspondence 246-248.

[38] J.Ziv (1964) Generation of optimal codes by a 'Pyramid-packing' argument: PGIT-10 253-255.

[39] L.Calabi (1964) On the minimal weight of binary group codes: PGIT-10 385-387.

[40] R.C.Bose and D.K.Ray-Chaudhuri (1960) On a class of error-correcting binary group codes: Information and Control 1

[41] W.W.Peterson (1960) Encoding and error-correction procedures for the Bose-Chaudhuri codes; PGIT-5 459-470

[42] 坂井利之，西尾英之助（1961）計算機によるハミング・コードの検討ならびに言語確率の統計：電気通信学会 通信方式専門委員会資料 1961-1-18

T. Sakai and H. Nishio (1961) Investigation of the Hamming Codes by means of the computer and the statistics of the Japanese language. IECEJ*

Communication system technical committee.

[43] 西尾 (1962) Minimum Distance Codes の構造：京都大学工学部電気関係教室研究談話会記録 NO. 105
H.Nishio (1962) Structures of Minimum Distance Codes.

[44] 坂井、西尾 (1963) 最短距離群符号の最大位数に関する計算結果： 昭和38年電気四学会連合大会予稿 27
T. Sakai and H.Nishio (1963) Computational Results of the maximal orders of the minimum distance group codes: Record of the 1963 Joint Convention of IECEJ*.

[45] 坂井、西尾、松尾壱郎 (1963) 真理値表で表現された論理関数の簡単化： 電気通信学会 トランスアクション「電子計算機の研究」 昭和38年 NO. 1 pp. 13-17
T. Sakai, H.Nishio and Ichiro Matsuo (1963) Simplification of the Boolean Functions represented by the truth table: Transaction "Study of Electronic Computer" 1962 NO 1, pp 13-17 IECEJ*

[46] 坂井、西尾 (1963) 試行錯誤による論理関数の近似：電気通信学会 インホメーション理論研究会資料 IT 63-7 1963-7-26
T.Sakai and H.Nishio (1963) An approximation method of logical functions by means of trial and error: IECEJ* Information theory technical committee IT 63-7

[47] 坂井. 西尾 (1963) 試行錯誤による論理関数の近似:
昭和38年 電気関係学会 関西支部連大 予稿 1-11

T.Sakai and H.Nishio (1963) An approximation of logical functions by means of trial and error.
-- on convergence of the Markov chain --
Record of the 1963 Fall Convention of IECEJ* Kansai branch.

[48] 坂井. 西尾 (1963) 試行錯誤による論理関数の近似:
電気通信学会誌 第46巻 11号 1705-1713頁

T.Sakai and H.Nishio (1963) An approximation method of logical functions by means of trial and error.
Journal of IECEJ*, 46, NO 11, pp 1705-1713

[49] 坂井. 西尾 (1963) 論理関数の近似: 昭和38年
情報処理学会 全国大会予稿 51-52頁

T.Sakai and H.Nishio (1963) Approximation of Boolean Functions: Record of 1963 Convention.
Information Processing Society of Japan pp51-52.

[50] Toshiyuki Sakai and Hidenosuke Nishio (1964):
An application of the group code theory to the approximation of Boolean Functions: ICMCI Tokyo Sept. 1964. Samaries of paper, Part 3, pp 163-164

[51] 坂井. 西尾 (1964) 論理関数の近似への群符号の応用:
電気通信学会 オートマトンと自動制御研究会 資料
AAC: 63-15 1964-1-26

IECEJ* = The Institute of Electrical Communication Engineers of Japan.

T.Sakai and H.Nishio (1964) An application of group code theory to the approximation of Boolean Functions: IECEJ* Automaton and automatic control technical committee. AAC- 63-15

[52] A.Feinstein (1957) Foundations of Inforiation Theory: McGraw Hill 136pp.

[53] 室賀, 喜安 (1956) 情報理論 (岩波応用数学講座)

[54] W.W.Peterson (1960) Error correcting codes: MIT Press 285pp.

[55] A.N.Kolmogoroff and W.M.Tichomirow (1960) Arbeiten zur Informationstheorie III --- ε-Entropie und ε-Kapazität von Mengen in Funktionalräumen: VEB Deutscher Verlag der Wissenschaften.

[56] W.Feller (1957) An introduction to probability theory and its applications 1: Second edition John Wiley 461pp.

[57] J.L.Doob (1953) Stochastic Processes: John Wiley 654pp.

[58] K.L.Chung (1960) Markov chains with stationary transition probabilities: Springer 273pp.

[59] J.Kemeny and J.Snell (1960) Finite Markov chains: van Nostrand 210pp.

[60] S.Hayashi (1961) Periodically interrupted electric circuits; Denkishioin 410pp.

[61] R.Frazer, W.Duncan and A.Coller (1955) Elementary matrices: Cambridge Univ. Press 416pp.

PART II

TRANSLATION OF PROGRAM INTO LOGICAL CIRCUIT

CHAPTER I

Introduction

Information appears in various concrete forms. Though information is an abstract object, it should take some real form, which is accepted by the observer. Since we think of the human communication, information must be accepted and utilized by man finally. At the intermediate stage, however, it takes several forms, so that communication may be efficient and not disturbed. The electric communication using the radio wave is the typical example. On the transmission channel, information has the form of the code, which is constructed from the original information, in order to prevent the noise. The Coding is a transformation of the form of information.

Thus transformation of information is the essential problem in the information processing.

Among many forms of information, there are those which exist in the natural world, such as the electromagnetic wave, the speech sound and so on, and those which are created by the human being and still abstract, such as the letter. The letter has also its concrete form, such as the printed character, when used as a tool of communication. But its essence is not the concrete form.

Every study of the natural science treats information

of the first form. Contrary to this the theory of information processing treats sometimes the second one, that is, the system of abstract objects. In other words, it is concerned with the structure of the system, whose each element has not much characteristics. In mathematics they study only such objects.

One of the most important systems of abstract things is the language. It is the system of letters. Its each letter or symbol is considered to be completely known, but its structure has not yet been clear. In studying the language we can not do without considering the automaton, which uses the language. In case of the natural language, however, the automaton is the human being. Since the structure of the human being is too complicated to formulate, it is quite difficult to study the natural language with respect to its accepter.

On the other hand, the digital computer has its language, or programming system, in order that the human being may communicate with it. In fact, if a man wants to do some logical work with the computer, he makes at first the program for it. The program can be regarded as the system to represent logical activities of the programmer. In recent years the theory of the program has been studied by many investigators. Mathematical linguistics plays an important role in this case.

The theory of programs and generally the theory of

algorithms have treated the set of basic symbols almost exclusively.

Here we should investigate them together with their acceptors or automata. Recent study of automata indicates clearly this tendency. This field, however, treats the elementary and abstract system of symbols as the input of the automaton. The automaton is also abstract. We treat the real programming system and the real logical circuit in this study.

The system of logical circuits is considered to be that of abstract things. The logical circuit has the real form, only when it is implemented as the electric circuit. It represents only the logical relationships among elements. Many investigators devoted their contributions to the study of logical circuit. There are two different phase of the study. One is the analysis and the other the design of the circuit. We are mainly concerned with the design problem here.

We take up transformation of the programming system to the system of logical circuit in this study. This is because both are essential in the modern information processing. Both are the concrete forms of information or algorithm, which represents the logical content of some job. Those who experienced once both programming and logical design, could see that there is very intimate

analogue between them. Of cource there are distinctive features of two systems. The most remarkable feature of the programming system is its flexibility in expressing the job and in computing it. Another feature is its linguistic structure. That of the logical circuit is its real time operation. The parallel processing is another feature. If we would transform them each other, we could have different forms of the same job and choose any one of them for the specific purpose.

One application of this concept is the automatic design of the logical circuit. The logical machine is so designed by the designer that it may execute what he intended as quickly and surely as possible with the least number of elements. For this the designer analyzes his job precisely in terms of the logical design technique such as the time flow chart, the theory of logical functions and so on. The quality of the design greatly depends on ability of the designer, because he carries out the work by his ingenuity sometimes, not doing always the routine work. This methodology has its merits and faults. We wish to give some other method to this problem. In recent years some investigators have contributed papers to the automatic design of the logical machine. Actually there are some practical computer programsfor this purpose.

Their methods are to express in the special computer

language the design factors such as the connection of elements, the timing relation and so on, and to utilize the computer for checking the logic and for getting the input equations of the logical elements.[3,5,6]

Our method is to transform the program directly to the logical circuit.

Its objective is to see,how the ordinary expression of the job by means of the program gives information about the logical design of the circuit,which executes the same job. In this respect the author does not claim that the study offers the optimal automatic design method, but intend to give the general methodology in this field, though he does not believe the end was reached.

The second objective is to serve the theory of programs or that of algorithms. Thus far many techniques have been developed to make the programming easier. The compiler is the representative of them. This is, however, concerned only with the symbolic processing of the programming system and not with the hardware directly. The author believes that the automaton itself should be considered in investigating the program, though the concrete method has not been reached.

One example of the programming theory is to establish the method for getting the optimal or the most economical program. It is said that the good program reduces the computation time much more than the fast logical element.

There are some efforts in this direction, though the results are limited. It is expected that the following theory will serve this field, too.[7]

One of the most difficult problems concerning the language is semantics. The programming language is considered to have semantics. For designing the good circuit by means of the linguistic approach, we should treat this problem. One example of semantic problems may be the choice of subroutines. In writing the program the programmer utilize subroutines often. The mechanism to determine, which part is to be the subroutine, is quite unknown. In designing the machine the same problem arises. Though this kind of semantics seems more tractable than that of the natural language, we have not formulated it effectively.

The definition of symbols appearing in the programming system in terms of the other system (language or other schematic descriptions) can be also regarded as semantics of the system. We define the symbol by means of the diagram of logical circuit. Another theory is necessary to solve the general problem concerning the definition of symbols.

The following chapters are devoted to the formulation and the concrete method of transformation (translation) of the program. Especially the algorithm to reduce the

number of logical steps of the program is presented. One way to reduce the computation time is to employ the concurrent operation, that is, such a machine that has many processing elements is faster than the ordinary machine, which has only one accummulater.

CHAPTER II

Formulation of Translation

§ 2.1 Meaning of translation

"Translation" is just "transformation" of the form of information into another form, but is the word, which is used especially for the language. The word "translation" is usually used for natural languages like "The translation of English into Germann". We mean here by this word translation of the program into the logical circuit. One of the characteristics to the translation in general is that the language, which is to be translated, as well as that to which it is translated, has linguistic structures, such as the syntax and the semantics. As is recognized among the investigators of the automaton, the program of the digital computer has also the very clear linguistic nature, especially the syntax. Contrary to this the logical circuit itself does not seem to have the syntactic property in its essence. But if we investigate the process of the logical design and the behavior of the circuit, it will be found that the system of hardwares could be treated like a linguistic system. In other words the system of the logical circuit and the system of the program belong to the same category of the system with respect to their linguistic structure.

So we can think of possibility of translation between them and in fact establish the concrete method of translation.

In translating some language it is required to keep the content of the sentence unchanged, while the other factors such as the form, the order of words and so on may change. Our method is to establish an algorithm of the translation which preserves the logical contents of the source program, while the execution time of it is reduced.

In general in translating natural languages we need some reference besides both languages. Especially definition of each word requires such a thing. For example the word "apple" could not be translated as "der Apfel", if there were not the concrete red fruit. This correspondence between the word and the object is recognized unconciously by man. In the study of translation little attention is paid to this relationship, because it is unnecessary for the practical problems to refer always to the background and the relation is too complex to formulate in the useful form. Therefore the correspondence between words in both languages like "apple" $\longleftrightarrow$ "der Apfel" is the starting point of translation. In our case the reference of translation is a logical machine, which is called the "kernel machine". Since the program is translated into the logical circuit, definition of each

word should be done in terms of the logical circuit.

Contrary to the natural language the logical circuit can be expressed precisely and it is rather easy to establish the algorithm of translation with reference to the kernel machine. The translation algorithm should depend only on the schematic description of the machine and the formal description of the programming system. In referring to the kernel machine, meaning of each symbol is defined. In this sense presentation of logical circuit could be called the semantics of the translation. The transformation algorithm of program described in CHAPTER IV will be the syntax of the translation.

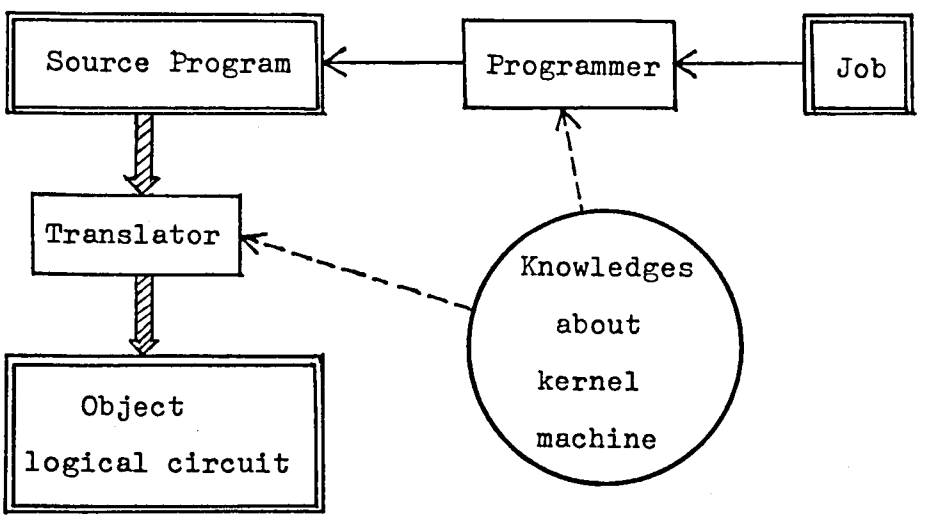
As is seen later we can not translate the program and design the useful logical circuit only by formal processing of symbols. This is the same situation as in the natural language. The logical element indicates its physical features which appear in different forms in each application. Definition of the single element does not give complete information about the phenomena which occur when many elements are connected each other. Therefore at the stage of translation where such situation arises, additional information should be given by man. In this sense the translation system can not be closed by itself.

§ 2.2 Method of translation

The concrete method of translation is explained here using the schematic diagram of Fig. 2.1.

The objective of translation is to obtain the (special purpose) logical machine, which executes the same job as the programmer intended. This machine is called the object machine, while the program, which is to be transformed, is called the source program. Here the special purpose machine means a circuit, which is not controlled by the stored program but has the definite control or timing circuit, nor has the addressable memory.

The translator is an transformation algorithm, which can not devise the new circuit by itself. That is, it



Method of translation

Fig. 2.1

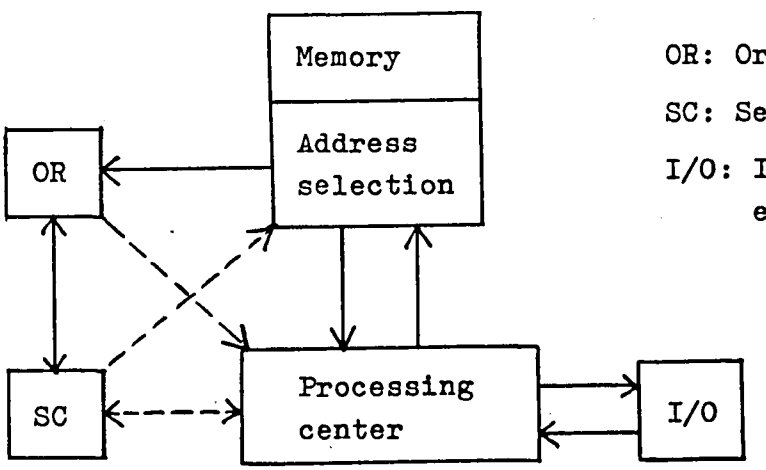
only "transforms" the form of information —the algorithm of some job in our case. The translator needs knowledges about the hardware, with which the object machine is to be constructed. All such informations are given from outside in terms of symbols or other formal representations. We give them in the form of register operation, because the ordinary logical machine is composed of several registers and logical elements connecting them.

In order to give the description of register operation, we used the "kernel machine". The kernal machine is the computer, for which the source program is programmed. The programmer programms his job utilizing knowledge about the behavior of the computer and expecting that the program and the computer will execute his intended job. In other words, the program, once it was written, describes the job completely, together with the description of the computer. Therefore in talking about the program, we can not do without the machine. Our translator needs also such knowledges. The kernel machine is described presicely in CHAPTER III using the concrete example.

The kernal machine (KM) is a universal computer, whose structure is quite well known and can be expressed easily, for example in terms of register operations. The general structure of KM is shown in Fig. 2.2. Its logical elements have simple and general properties so

that the construction of more complicated logical circuits from them may be straight-forward. We employed the transistor delay element and the diode logical element, because they need no caution about the logical timing relationship among elements, contrary to the multiphase elements such as the parametron. As to KM, it is assumed to be clear, which part of KM corresponds to each instruction of the programming system. In our translation, this correspondence is regarded as definition of the instruction.

The schematic description of KM is, however, not sufficient sometimes for giving all necessary informations about the logical design. In the course of translation extra knowledges are required. They differ from case to case and could not be formulated beforehand. They concern in general with the physical limitation on the elements and the technical and economical requirements on the construction of circuits. For instance, the diode logic of many stages can not be implemented because of



OR: Order Register
SC: Sequence Counter
I/O: Input Output equipments

Kernel Machine

Fig. 2.2

the time delay and the distortion of wave forms. Therefore the translation algorithm can not be general and depends on the chosen KM, especially on its elements.

The source program P is written in the machine language, not in the higher level language such as the symbolic or the automatic programming system. This is because, otherwise the translation would become too complicated to formulate. The program written in the machine language corresponds directly to the series of basic logical operations and therefore to the series of the logical circuits for them.

The format of the program is not essential. We formulate the programming system in § 3.2 as the system of strings of words for the sake of simple representation. In writing the program actually, however, this is not suitable and it is convenient to write it symbolically and the program in many lines as is done usually. In this study examples are given in this form.

Let's make it clear here, that what is transformed is not the program only, but the program plus a part of KM. Fig. 2.3 illustrates the parts of KM which are involved in translation. It is difficult in general to classify the memory into the program memory, the working memory and the data memory, since the memory is used for various purposes during the execution. We restrict,

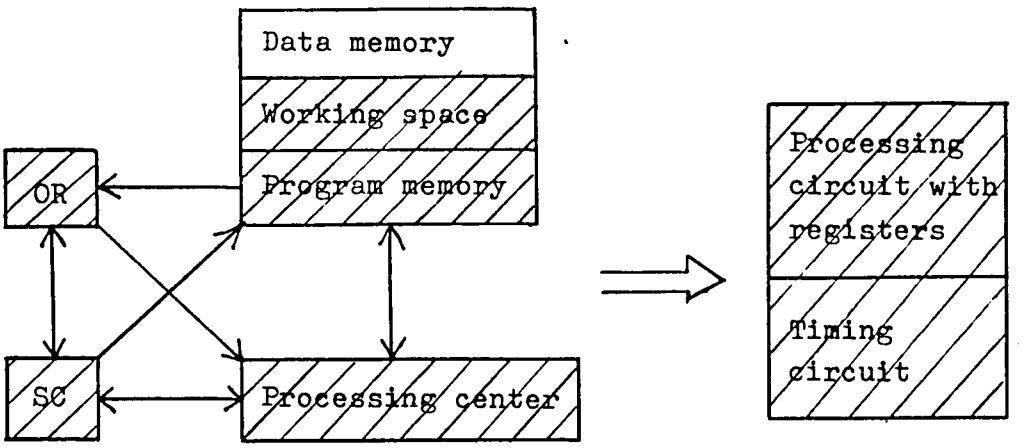
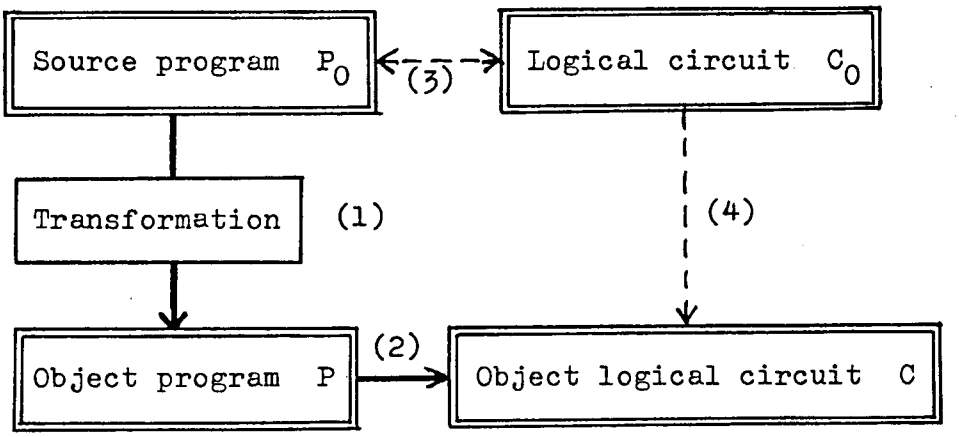


Fig. 2.3 Parts which are translated

therefore, the source program to a program such that the memory allocation is fixed. Roughly speaking the program memory, the sequence counter (SC) and the order register (OR) are translated to the timing circuit of the object machine, while the accummulator (AC), OR and the working memory space are to the operation part. Thus the object machine is composed of two parts. The input or output operation is treated as the register operation between the input-output register and AC.

The translator consists of two parts, one giving the correspondence between the symbolism of the programming language and the schematic description of the logical circuit, the other being the main part, which transforms the source program into another program. Fig. 2.4 shows it. The correspondence (3) is definition of the instruction and (2) is description of the object machine. Both



Routes of translation

Fig. 2.4

correspondences are obvious and need not analysis. If the translator is expressed as the program of the digital computer, then the representation of logical circuit in the computer gives them. In the route of translation from P_0 to C through (1) and (2), there is the intermediate stage P. P is introduced to formulate the concurrent operation of many instructions. The concurrent operation reduces in general the execution time of the object machine. P is a kind of program, which is not executed by any real computer. Or we have not the machine which executes P. The programming system of P is described in § 3.2 and CHAPTER IV.

Transformation (1) is the main subject of the translation. The algorithm of it preserves the logical content of the source program, while it reduces the number of logical steps. Also, if possible, it is required to

reduce the number of logical elements which are necessary to construct the object machine. P is the object program of the transformation algorithm (1).

In the theory of algorithms there is the general problem concerning the equivalent transformation. The theme is to obtain the algorithm which preserves the equivalence relation of a certain given algorithm and minimizes (or maximizes) a certain value attached to it. Our transformation is just an example of this problem.

In CHAPTER IV we use the term "transformation" of programs to indicate the transformation (1).

The other route of translation in Fig. 2.4 goes through the broken line, $P_0 \rightarrow C_0 \rightarrow C$. This route could not be formulated because transformation (4) of logical circuits is unknown. It is not so preferable to use the computer to process the geometric pattern directly in its form and some symbolic representation of it is necessary.

Finally translation in the reverse direction is conceivable. But the transformation algorithm (1) is not reversible and we do not treat this problem here.

§ 2.3 Limit of translation

There are two kinds of limits to our method of translation. The first one is the essential limit, caused by the general property of the source program, and connot be removed easily. Straight-forwardly speaking, such a pro-

gram that has the self-modifying routine can not be translated. The self-modifying program modifies some instructions of its own in the course of execution, so that we can not completely know the sequence and the operations of the programs before execution. If we dare to translate such a program, the self-modifying or self-organizing logical circuit would be required. Since we think of the ordinary logical design here, so this belongs to another field of the study.

The second limit arises from the practical problems. As is seen later we make each address symbol appearing in the program correspond to the register. Therefore if the program used many memory registers, the object machine would have also many registers, though the number of them is reduced by the algorithm. It is not practical to design the machine which requires so many elements. Therefore we exclude also such jobs that use many memory registers. In the following analysis, we treat the jobs, which process the information contained in the accumulator and use the memory as the intermediate working space. They require only a few input data and their results are placed at the accumulator.

Since our algorithm does not take into account the engineering intuition, there is possibility to design a non-sense object machine. This is mainly because the choice of KM is not always suitable. If KM has not

suitable instructions for the considered job, the object machine is badly designed. We have no algorithm to exclude such a non-sense design. The quality of the sourse program affects also very much that of the designed object machine. We need therefore so called pre-edit and post-edit by man, in order to obtain good machines.

CHAPTER III

Descriptions of Kernel Machine and Programming System

As is stated in the previous chapter the translation system is based on the kernel machine, which has the suitable programming system. In this chapter the kernel machine and its programming system, which we defined for the study, are described.

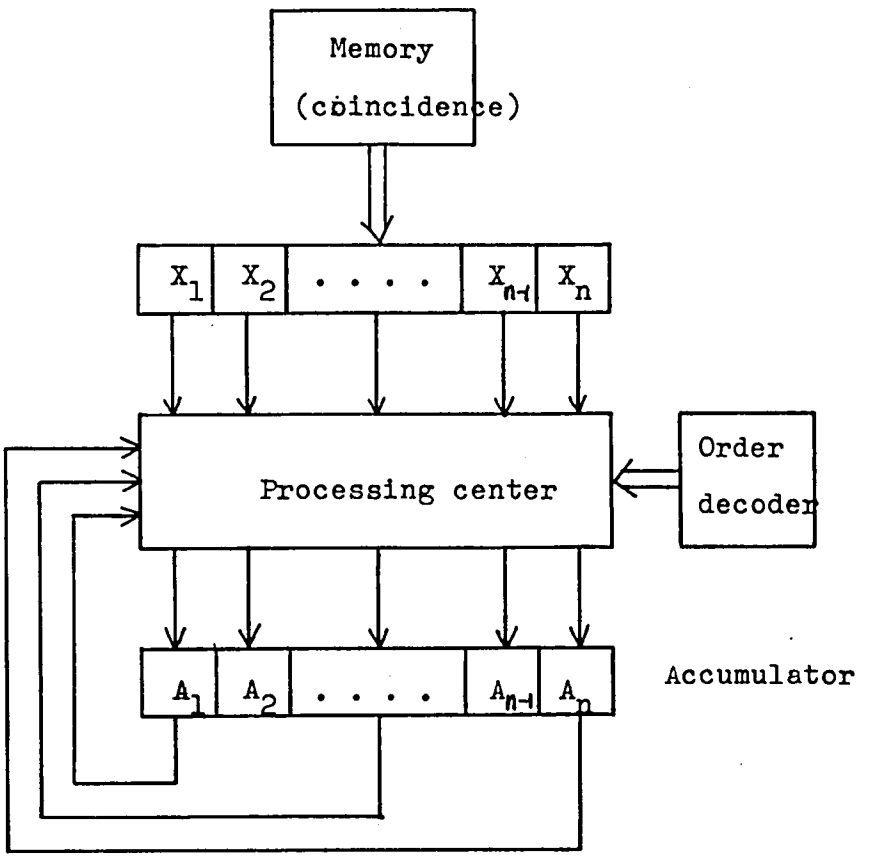
For the formal representation of KM, we utilize the illustration by the schematic diagram and the ordinary logical circuit. The programming system is formulated so as to serve the transformation algorithm given in the next chapter.

It is not claimed that the descriptions specify the system completely. It may be sufficient to give the understanding about what system is going to be treated.

§ 3.1 Kernel machine

Kernel machine is the basis of translation and should be specified precisely. But as is seen from the previous chapter, all parts of KM are not involved in translation. For instance the memory and its coincidence circuit may be assumed arbitrary. The I/O equipments are also not essential. We need only to specify the processing center and the accummulator.

We choose the concrete KM, of which the processing center is indicated in Fig. 3.1, and investigate trans-



Processing center of KM

Fig. 3.1

lation using it exclusively. This KM was devised conceptually only for the purpose of translation and is not the real machine, but clearly can be implemented easily if wanted, because it does not contain any unrealizable element. In the following explanation this special KM is referred to always.

The necessary properties, which KM should have, are as follows:

(i) The general purpose computer

KM should be general in the sense that any logical operation can be done between any pair of bits of memory or register without changing the other bits. For this, our KM has the instruction set as indicated below:

Instruction set:

AND X EOR X OR X NOT RSH i LSH i

STO X DRW X RIN WRT JMP X JNZ X HALT

where X is the general symbol for the memory register and i is the positive integer.

Each instruction is precisely explained later by using the logical circuit of the processing center.

If there were not the shift operator, operation between the corresponding bits of AC and of the memory register would be impossible.

Generality is required, of course, in order to guarantee the generality of the job which is to be translated. If the kinds of the source program are limited, then the generality is unnecessary. But in this case the object machine is also limited.

As is supposed from the instruction set, we think of the jobs which are not of direct use but are the basic logical operations on binary words such as counting the number of 1s, addition of binary numbers and so on. Since the instruction set does not contain the higher operation such as Addition and Multiplication, such a program as computes SIN X or $\sqrt{X}$ is not programmable easily for our KM.

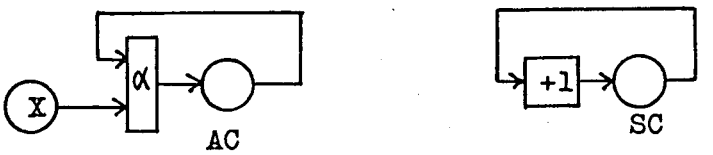
We do not insist upon the minimality of the instruction set. There may be redundant instructions in order that the instruction set may be general. The set was chosed from the practical point of view.

(ii) The binary parallel synchronous machine

KM is the synchronous machine, each instruction of which is executed in a unit of clock time. Operation is done in parallel for each bit of a word. A word consists of n bits, where n is a fixed positive integer. For the practical KM, n is larger than 20. For our study the specific value of n is not essential. Fig. 3.1 shows the processing center of KM. Every information

(i) Binary operator

α(AC, X) → AC and SC+1 → SC



(ii) Unary operator

β(AC) → AC and SC+1 → SC

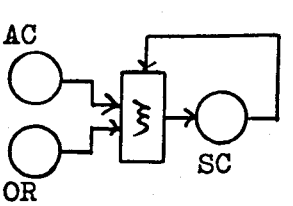
(iii) Transfer operator

•(AC) → X and SC+1 → SC

or •(X) → AC

(iv) Sequence operator

ξ_{AC}(X, SC) → SC



: Register

: Operation circuit

Schematic representation

Fig. 3.2

is expressed, transmitted and processed in the form of the n-bit binary word en bloc.

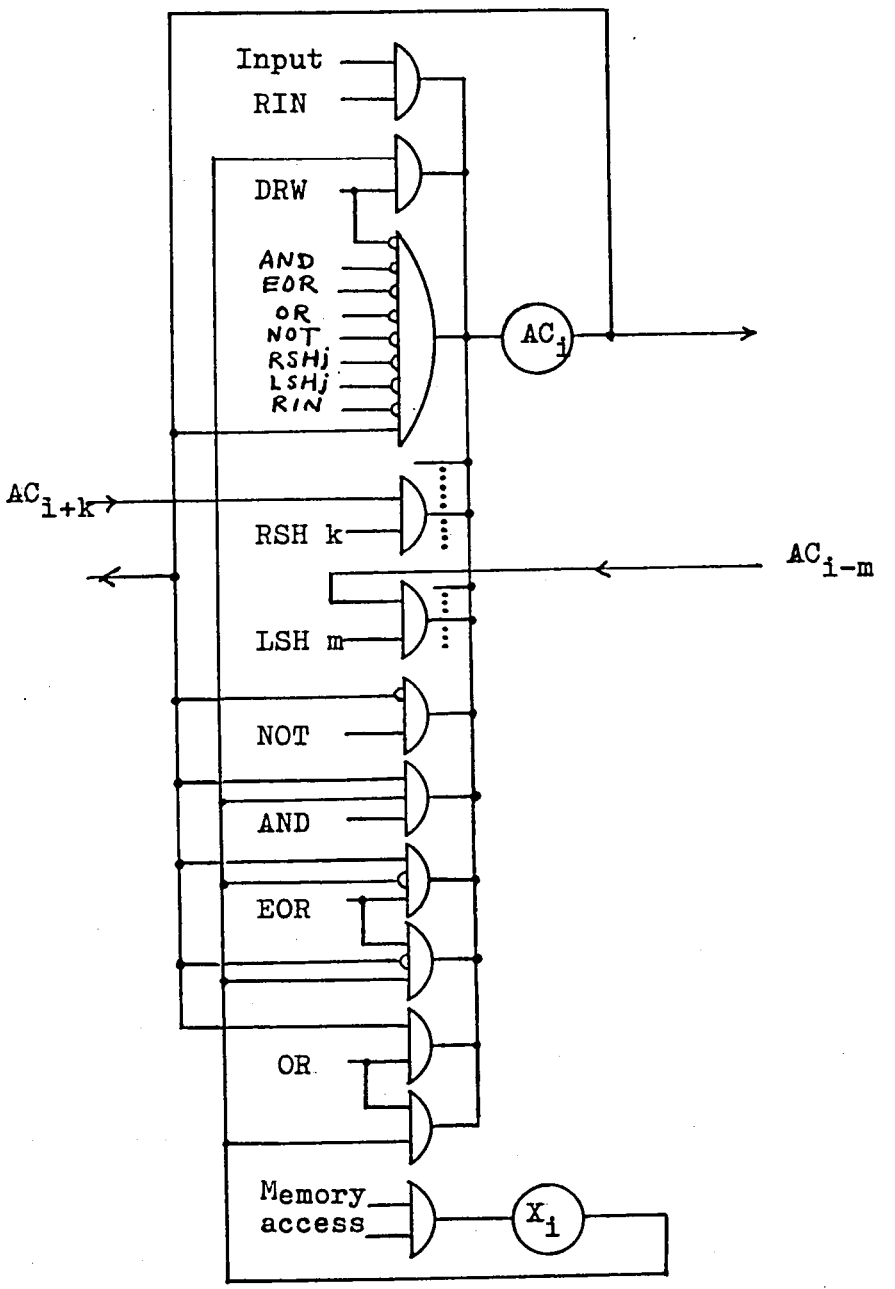
The operation is made only between AC and the memory register or on AC. The former is called the binary operation and the latter the unary operation. Results of both operations are placed in AC.

In general the binary operation is the set of n 2n-variable Boolean functions and the unary operation is the set of n n-variable Boolean functions. They define the input equation of each bit of AC.

Fig. 3.2 shows the operations schematically. The circle means the n-bit register, the rectangle means the operation part and the line transmits n-bit information in parallel. We do not use the figure indicating the logical circuit but utilize this diagram in the study of translation.

Fig. 3.3 shows the i-th bit of AC and its operation part. For the 1-st and the n-th bit of AC some modifications are necessary to Fig. 3.3.

The name of the instruction indicated there means the output of the instruction decoder, which is omitted from the illustration. The timing clock is not expressed explicitely but enters each gate so that every operation is finished in a clock time. Since there is no other counter than SC, there is the independent gate for each shift operation.



The i-th bit of Processing Center

Fig. 3.3

X_i is the general symbol for the memory register, which is selected by the address part. Since the logical content of each instruction is defined completely by Fig. 3.3, we explain its meaning roughly.

AND X, OR X, EOR X are binary operations of X and AC.

NOT is negation of each bit of AC.

By RSH i or LSH i, AC is shifted by i bits to the right or to the left respectively.

STO X and DRW X are complementary operations each other. STO X stores the content of AC to X and DRW X draws the content of X to AC.

RIN and WRT are input and output operations, by which a word of data is transfered between AC and the I/O equipments. These operations are also assumed to be finished in a clock time.

JMP X is the ordinary sequence operation to X. JNZ X modifies the program sequence to X, if every bits of AC are zero. HALT stops the KM. The last three operations concern SC and are not indicated in Fig.3.3.

(iii) The ideal logical element

For the basic elements of KM we assume the ideal elements which have the following properties.

The logical elements, i.e. the delay element and the operation element, should be simple in its logical structure. Any output of the delay element should be able to

go to arbitrary operation elements, without considering the timing relation. In this respect such elements as parametron are not suitable for the formulation of translation.

The delay and the operation elements should be separated. There are such elements as have the functions of operation and delay in connection. For these elements we can not insert any other element between the operation and the delay elements. Therefore they are not suitable for constructing the complicated logical circuit by our algorithm.

The time delay is assumed to be only due to the delay element so that the operation element operates instantaneously.

The number of elements (the number of fan-outs) which can be connected to one delay element in parallel is limited for the practical elements. But in this study, it is assumed that this number may be arbitrarily large. It is easy to restrict the number in each practical design problem by modifying the transformation algorithm. The same condition is assumed for the number of operation elements which can be connected in cascade without any delay element among them. If a delay element is inserted at a branch of logical circuits, then the timing of the operation may be disturbed. In the real logical design, the designer adjusts the timing by inserting dummy delay element at the other branches to compensate it. Such a

technique is not essential in our case, so we assume the ideal case. These assumptions were not met with difficulty in the examples given later, because the number of operations contained in each compound operation is not so large. At most three or four stages of AND-OR gate logic are constructed by the algorithm.

Choice of kernel machine

Translation depends greately on the choice of KM, in particular that of the instruction set. In writing the program in general, if the instruction set is not suitable for the job, many program steps, and therefore much computation time, are necessary. In our case such dependence does not exists directly, because the source program is transformed by the reduction algorithm of program steps. But there are always unfavorable jobs for the specific KM. For example for KM of Fig. 3.3, the program which inverts the order of the content of AC, i.e. $(X_1, X_2, \ldots, X_n) \longrightarrow (X_n, X_{n-1}, \ldots, X_1)$, requires many steps. If KM had an instruction for such an operation, the program is trivial. By the way, this operation is quite easily implemented.

Choice of the instruction set has been one of the biggest problems of logical design of the computer. This can not, however, be solved generally.

Systematic research of the instruction set is impossible for our study, too. From the restrictions on KM

stated thus far, we can not choose such instructions as ADD or MULT which might require many logical steps to execute, but those instructions are to be designed by translation of the source program for them.

§ 3.2 Programming system

There are various ways to describe the programming system, ranging from the quite formal representation to the vague verbal exposition. For the theory of programs or algorithms the formal representation of the system is necessary but for the practical programming the precise formalism is not always required. It is not our intention here to give the complete description of the system, but it is sufficient to indicate what type of the programming system we are going to employ.

In transforming the program the main interest goes to the sequence of it, not to the concrete operations. This is because the program should be always the general algorithm, that is, the same program is used for various input data. So we should not take into account the concrete data or the contents of registers. For example, the content of AC controls the sequence of the program if it has the conditional jump with condition on AC. But the instance when the condition is satisfied can not be determined beforehand. Therefore we must neglect the concrete data. The timing circuit of the object machine is designed by following the execution of the source pro-

gram. Execution of the program should be therefore analyzed precisely. We do it by classifying the instructions into the normal and the sequencing operators.

For the non-self-modifying program the execution can be specified completely by the form of program, but for the self-modifying program, its sequence can not be determined in general. For expressing the program sequence the sequence diagram is used in this paper.

The linguistic description of the programming system consists of "definition of the used symbols" in a non-linguistic way and "the structural description of the syntax". The former is called sometimes "semantics" of the system and is given by the schematic diagram of the logical circuit shown in Fig. 3.2.

§ 3.2.1 Form of program

The program consists of many instructions or words. Each word consists of the location part, the operation part and address part. At the location and the address parts the register symbols is written. We use the symbolic representation of the memory register. At the operation part is written the operator symbol. The general form of the word is one of the following two forms,

(i) a α

(ii) a α X

where α is the operation symbol and a and X are the register symbols. The operator NOT, RIN and WRT are of type (i), while AND, OR and so on are of type (ii). RSH i and LSH i are considered to be of type (i).

In translation, X is translated to the n-bit register of the object machine, while α is translated to the logical operation part. Each α is defined by the diagram of Fig. 3.2. The location part a is translated into the timing circuit, after analyzed by the sequence diagram.

Fig. 3.4A is an example of the program. It is a program, which counts the number of ones in AC and place the result in AC in the binary coded decimal form. This example is called TAKE WEIGHT.

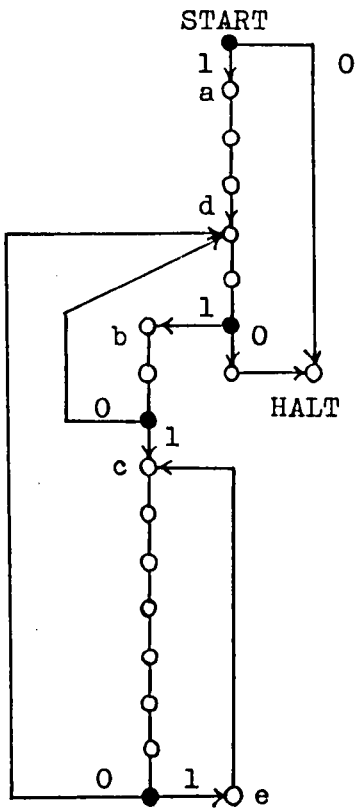
§ 3.2.2 Program sequence

The program does represent the job completely, if it is not executed.

The complete description of the job, which the programmer intended, is obtained only after the execution of the program. Therefore we should investigate, how the program is executed.

But it is seen that the behavior of the general program can not be specified by finite number of steps. For example, if the program contains the "Loop", there may be infinitely many behaviors.

```
START   JNZ   a
HALT    HALT
a       STO   D
        RSH   n-1
        STO   C
d       DRW   D
        LSH   1
        JNZ   b
        DRW   C
        JMP   HALT
b       STO   D
        RSH   n-1
        JNZ   c
        JMP   d
c       STO   A
        DRW   C
        STO   B
        EOR   A
        STO   C
        DRW   B
        AND   A
        JNZ   e
        JMP   d
e       LSH   1
        JMP   c
```



Program of TAKE WEIGHT

Fig. 3.4A

Sequence diagram

Fig. 3.4B

If the program is self-modifying, the program sequence itself is determined according to the execution.

There is such a program that itself generates the instruction, which it executes later.

In our translation, we treat only the symbols which appear on the program sheet and exclude such an ambiguous program.

In other words, we treat the program, of which the sequence is completely specified only by the normal sequence and the sequence operator before execution. We assume also the program, which has been loaded on the computer, is the same as the written program.

The sequence is represented by the sequence diagram. Fig. 3.4B is the example of the sequence diagram for the program TAKE WEIGHT indicated at the same page.

The white circle means the normal operator and the black point the conditional jump. The unconditional jump is neglected in writing the diagram. The diagram is constructed by the following algorithm: Write the starting point at first. Then following the normal operators write the successive circles and connect them by line. If the conditional jump appears, then write the black point and make a two-way branch. By following each branch, continue the same procedure, until the branch is connected to some point of the diagram or ends with the HALT instruction.

At the branching point, the symbol 1 is attached to the branch, for which the condition was satisfied, and the symbol 0 to the branch, for which the condition was not satisfied.

In CHAPTER IV we use the word "branch" in this sense.

The sequence diagram is considered to be the state transition diagram of the timing circuit of the object machine.

CHAPTER IV

Transformation of Source Program

As is formulated in CHAPTER II, it is the transformation of program that can be treated rigorously.

At first, since the register operation is the basis of representing the logical circuit in our system, we rewrite the source program in this notation. Then we transform it by the algorithm to get the object program. The system of object programs has the instructions which are obtained by extending the meaning of the original instructions.

The main alteration of the system is that the concurrent operation and the compound operation have been introduced. The concurrent operation, called also the parallel operation sometimes, means to execute many instructions in a clock time. The compound operation is an logical operation which is the function of the original operations. We will call "compound operator" such operators that contain the concurrent and the compound operators.

The transformation algorithm gives the rule to construct the compound operator which is equivalent to a portion of the source program. It was so chosen that the number of steps of the object program may be as small as possible. And then, if possible, it is tried to re-

duce the number of registers which appear in the object program. These both criteria are complementary each other. We prefer the reduction of program steps to that of number of register symbols in our algorithm. The general algorithm is given as well as some examples.

§ 4.1 Generalization of Operation

We assumed the general logical elements for KM. Here we extend each operator to the general registers. For example, NOT was the negation operator of the content of AC, the result being placed in AC. There was no reason, however, to restrict the operand to AC. Here we consider the general negation operator which operates on the arbitrary n-bit register and whose result is also placed in the arbitrary n-bit register.

In defining the new interpretation of each instruction, we use the schematic diagram of Fig. 3.2.

The instruction can be represented by the register operation formula as follows.

Let the content of the register be denoted by the same symbol as the register itself.

(i) Binary operator

$$a \ \alpha \ X \equiv a \ \alpha(X) \rightarrow AC \text{ and } SC+1 \rightarrow SC$$

(ii) Unary operator

$$a \ \beta \equiv a \ \beta(X) \rightarrow AC \text{ and } SC+1 \rightarrow SC$$

(iii) Transfer operator .

$$a\ \gamma X \equiv a\ \bullet(X) \rightarrow AC \text{ and } SC+1 \rightarrow SC$$
$$\text{or } a\ \bullet(AC) \rightarrow X \text{ and } SC+1 \rightarrow SC$$

where $\bullet(X) \rightarrow AC$ means the content of X is transferred to AC without any operation on it.

(iv) Sequence operator

$$a\ \xi X \equiv a\ \xi(X) \rightarrow SC$$
$$\text{or } a\ \xi_A(X,SC) \rightarrow SC$$

where $\xi(X)$ means X itself and $\xi_A(X,SC)$ means that if the condition on AC is satisfied, the content of SC is changed to X and otherwise SC+1 is placed in SC.

Generalization means the following scheme. Let X, Y and Z be the general registers of n-bits.

(i) Binary operator

$$a\ \alpha(X,Y) \rightarrow Z \quad \text{and} \quad SC+1 \rightarrow SC$$

(ii) Unary operator

$$a\ \beta(X) \rightarrow Z \quad \text{and} \quad SC+1 \rightarrow SC$$

(iii) Transfer operator

$$a\ \bullet(X) \rightarrow Z \quad \text{and} \quad SC+1 \rightarrow SC$$

(iv) Sequence operator

$$a \quad \Xi_X(Y,SC) \rightarrow SC$$

where Ξ_X means that the condition of branching is concerned with X.

By this generalization, AC has lost its speciality as the center of processing. We treat all register symbols equally. Note that SC is kept always unaltered and is the register for controlling the program sequence.

The above generalization is allowed naturally, since it requires no constraints on the physical element. But the extension of each register, for example, to m-bit register ($m > n$) may be not allowed without consideration of the engineering problems, such as the stability of physical elements.

§ 4.2 Concurrent operation and compound operation

Concurrent operator[4]

Concurrent operation is defined in order to reduce the number of program steps. Suppose, for example, there is the portion of program as follows:

```
      .  .  .
      .  .  .
a   STO  X
b   AND  Y
      .  .  .
```

Then both operations can be executed concurrently or in parallel. We express this situation by the new symbol as follows:

```
      .  .  .
      .  .  .
a  STO  X  ∩  AND  Y
      .  .  .
      .  .  .
```

In general the concurrent operator is defined as an operator which was the following form.

$$a \quad \alpha_1 A_1 \cap \alpha_2 A_2 \cap \alpha_3 A_3 \cap \cdots \cap \beta_1 \cap \beta_2 \cdots,$$

where α_i and β_i are normal operator symbols and A_i is register symbol.

Compound operator

We are met with the following situation often in the course of programming:

```
      .  .  .
      .  .  .
a  AND  X
b  OR   Y
      .  .  .
      .  .  .
```

This is the cascade of two logical operations, which is illustrated by the register operation as follows:

```
      .  .  .
      .  .  .
a   AND(A,X) ⟶ A
b   OR (A,Y) ⟶ A
      .  .  .
      .  .  .
```

The content of AC at the step b is the result of the imediately previous step a. We can combine two operations in a compound operation, which has the form:

```
      .  .  .
      .  .  .
a   OR( AND(A,X), Y ) ⟶ A
      .  .  .
      .  .  .
```

The meaning of this formula may be clear. This is a kind of the concurrent operation.

Schematic representation

The sequence of two instructions of the source program is represented by the schematic diagram of the logical circuit as follows:

```
      .  .  .
      .  .  .
a1  α1(A,X) ⟶ A
a2  α2(A,Y) ⟶ A
      .  .  .
      .  .  .
```

Fig. 4.1A

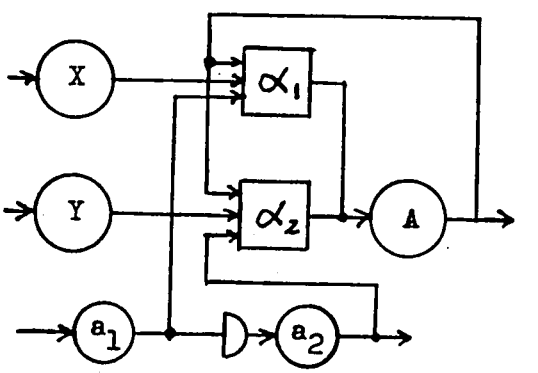


Fig. 4.1B

The whole source program is also represented by such a scheme. The circuit for a_1, a_2 is the timing circuit. It has the form of ring counter, such that one and only one pulse exists in a delay element and the output of it triggers the gate of the corresponding operation circuit.

The concurrent operation in the generalized notation is represented by Fig. 4.2.

$$
\begin{array}{l}
\vdots \\
a_1 \quad \alpha_1(X_1, Y_1) \rightarrow Z_1 \cap \alpha_2(X_2, Y_2) \rightarrow Z_2 \\
\vdots
\end{array}
$$

Fig. 4.2A Concurrent operation

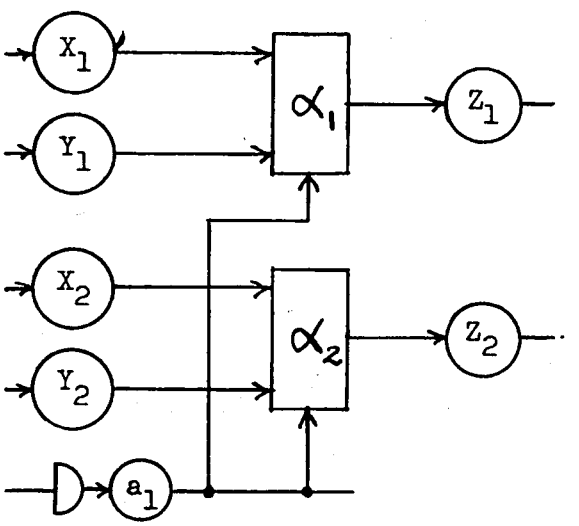


Fig. 4.2B Concurrent operation

The compound operation is represented by Fig. 4.3.

$$\begin{array}{c} \cdot\ \cdot\ \cdot \\ \cdot\ \cdot\ \cdot \end{array}$$

$$a_1\ \alpha_1(\ \alpha_2(X_2,Y_2),\ \alpha_3(X_3,Y_3)\ ) \rightarrow Z_1$$

$$\begin{array}{c} \cdot\ \cdot\ \cdot \\ \cdot\ \cdot\ \cdot \end{array}$$

Compound operation

Fig. 4.3A

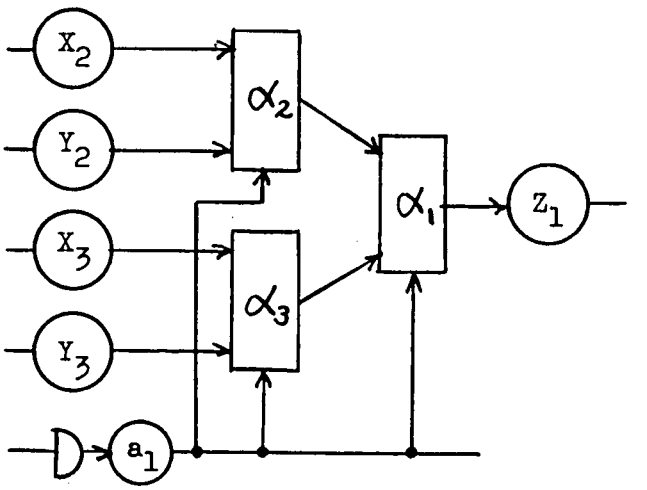


Fig. 4.3B Compound operation

These correspondences are generalized to the concurrent operation of many steps, or to the compound operation of many factors.

We use the following notations to avoid the nuisance symbols.

$\overline{X}$: the ordered set of finite resister symbols such as:

$\overline{X} = \{ X_1, X_2, \ldots, X_k \}$.

$\bar{\alpha}(\bar{X})$: the compound logical function on $\bar{X}$ such as:

for $\bar{X} = \{X_1, X_2, X_3, X_4\}$,

$$\bar{\alpha}(\bar{X}) = \alpha_1 \{\alpha_2(X_1, X_2), \alpha_3(X_3, X_4)\},$$

where α_i is the original normal operation symbol.

$\bar{\alpha}(\bar{X}) \rightarrow \bar{Z}$: The function value of $\bar{\alpha}$ for $\bar{X}$ is placed in all registers contained in $\bar{Z}$. This is the general form of the single compound operator. The registers which appear in $\bar{X}$ are called the operand registers and those in $\bar{Z}$ the destination registers.

Then we have the following definition of the concurrent compound operator.

Definition:

The concurrent compound operator is a series of compound operators connected by the symbol $\bigcap$, for which the destination registers of each compound operator are different. The ordinary operator is called the simple operator.

The general form of the concurrent compound operator is as follows:

$$a \quad \bar{\alpha}_1(\bar{X}_1) \rightarrow \bar{Z}_1 \bigcap \bar{\alpha}_2(\bar{X}_2) \rightarrow \bar{Z}_2 \bigcap \cdots \bigcap \bar{\alpha}_k(\bar{X}_k) \rightarrow \bar{Z}_k$$

We abbreviate the notation as follows:

$$a \quad \bigcap_{i=1}^{k} \bar{\alpha}_i(\bar{X}_i) \rightarrow \bar{Z}_i$$

The condition on the destination registers stated in definition is expressed set-theoretically as follows:

$$\bar{Z}_i \cap \bar{Z}_j = \phi \qquad (i \neq j)$$

where $\cap$ means the intersection of sets and ϕ is the empty set. This condition is required for the sake of consistency of concurrent operations.

Concurrent operation of sequence operator

Thus far the concurrent and the compound operations have been limited to the normal operators. Here we can think of those sequence operators. For example the pair of operators as follows can be executed concurrently.

```
      .   .   .
      .   .   .
a   AND   X
b   JNZ   Y
      .   .   .
      .   .   .
```

In fact, the following compound operator is consistent.

```
      .   .   .
      .   .   .
a  AND(A,X)→X  ∩  ξ_AND(A,X)(Y,SC)→SC
      .   .   .
      .   .   .
```

where $\xi_{AND(A,X)}$ means the condition on the result AND(A,X).

The general form of the concurrent compound operator is therefore as follows:

$$\bigcap_{i=1}^{k} \bar{\alpha}_i(\bar{X}_i) \longrightarrow \bar{Z}_i \cap \xi_{f(\cup \bar{X}_i)}(X,SC) \longrightarrow SC,$$

where $\cup \bar{X}_i$ is the set union of $\bar{X}_i$ and f is a logical function of $\cup \bar{X}_i$ and ξ operates on its function value so that if all bits are zero, then $SC+1 \longrightarrow SC$ and otherwise $X \rightarrow SC$.

We studied also the following situation:

. . .
. . .

a $\xi_A(X,SC) \rightarrow SC$

b $\xi_B(Y,SC) \rightarrow SC$

. . .
. . .

This is the branching of the program sequence into three alternatives according to the contents of A and B. If the condition of A is satisfied, then $X \rightarrow SC$, if not and the condition B is satisfied, then $Y \rightarrow SC$, and finally if either condition is not satisfied, then $SC+1 \rightarrow SC$. We execute both operations concurrently by constructing the logical circuit for SC. But this process could not be formulated completely. Therefore we do not enter this problem here.

§ 4.3 Timing circuit

As indicated previously the sequence diagram corresponds to the timing circuit of the object machine. The diagram is considered to be a state diagram of automaton with the input given by the condition of AC. Therefore we can design the timing circuit by the state asignment theory of the finite automaton. In practical cases the binary coding is employed for the state asignment. In fact, SC of KM, as well as the sequence counter of the ordinary computer, is the binary counter. This asignment is not optimal for specific program sequences, but has generality and can be easily designed.

We do not, however, encode the state by the binary counter, but use one bit of flip-flop for each state. Therefore the same number of flip-flops is used for the timing circuit as that of the circles of the sequence diagram.

This kind of the circuit is the direct counter-part of the program sequence, while the circuit, which is obtained through the state asignment, has lost the direct correspondence with it.

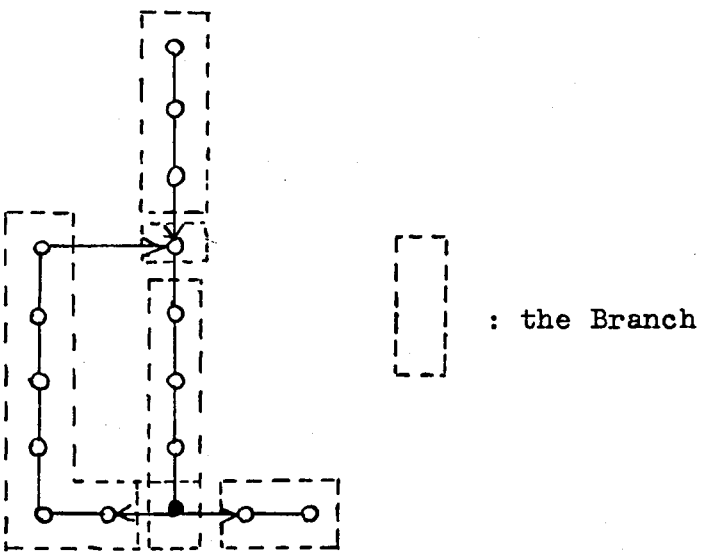
An example of such timing circuits is given in Fig. 4.10, which corresponds to the object program. Like the ring counter, one and only one pulse exists in the circuit and goes around the circuit giving the appropriate timing pulsesto the operation part.

§ 4.4 Transformation algorithm

§ 4.4.1 Extraction of branches

The program is not transformed as the whole, but treated branch by branch. A branch of the program is the portion, which has no conditional jump, nor the instruction having more than one inputs to it. (See, Fig. 4.4.)

If we are to transform the instructions which are contained in different branches, we must consider the condition of AC for many situations according to the previous sequences the program took. Though we established some rules to treat the transformation over the limit of branch, they are hardly general and omitted



Branches of Sequence diagram

Fig. 4.4

here. Since we transform only branches, the loop structure of the source program is kept unchanged by transformation. That is, the sequence diagram has the same shape, while the number of white circles contained in each branch is reduced. In some special cases a loop shrinks so that it contains only one concurrent compound operator. Note that even in this case the loop can not be eliminated.

The main feature of the sequence diagram of the program is that it has loops in general. There are, of course, those programs which have no loop, but from several reasons the programmer uses loops often. The ordinary program is constructed recursively and the loop is necessary.

It is supposed sometimes that the better machine could be designed by altering the structure of the sequence diagram itself. But it has been impossible to establish the general algorithm to do so. Therefore we investigate the transformation branch by branch.

Though the branch is extracted by means of the sequence diagram described in § 3.2, it can be detected quite easily by the symbolic processing of the source program. The conditional jump is regarded as the last step of a branch.

We apply the algorithm described in the following sections to every branches independently and connect them

after transformation to obtain the object program.

§ 4.4.2 Rule of applying transformation formulas

The algorithm consists of the set of transformation formulas listed in § 4.4.3 and the application rule of them, which is given in this section.

The general form of the transformation formula is as follows:

$$\left.\begin{matrix} a \\ b \end{matrix}\right\} \Rightarrow c$$

where a is the concurrent compound operator (the simple operator included), b is the simple operator and c is the new concurrent compound operator constructed from a and b. The formula is applied successively for the pair of operators from the top to the end of the branch.

In order to show the rule of application of formulas, we use the notation (a, b) for the operator c. Then two lines of operators are transformed to a line, so that

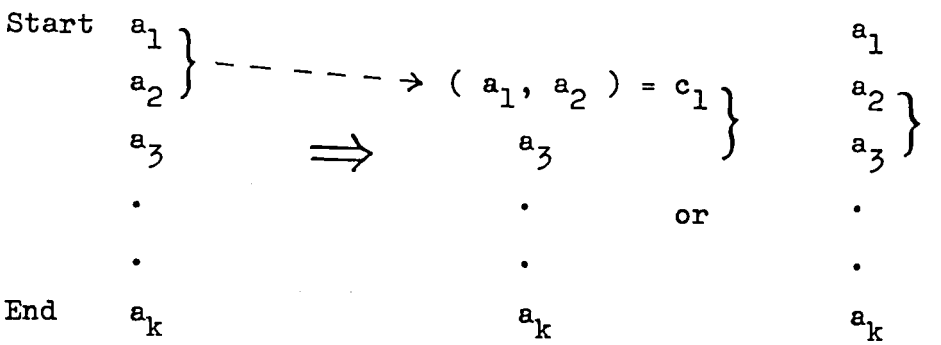
$$\left.\begin{matrix} a \\ b \end{matrix}\right\} \Rightarrow (a, b).$$

Let $a_1, a_2, \ldots, a_k$ be the instructions of the branch which is to be transformed. a_1 is the starting point and a_k is the end point of the branch. a_i is the simple operator.

The branch is transformed according to the following

rule:

(i) Take the pair of operators a_1 and a_2. If there is a formula in the list, whose left hand elements are identical with them, rewrite the pair by the concurrent compound operator which is the right hand of the formula. If there is not, leave a_1 as it is and go to the next step. (See, Fig. 4.5)



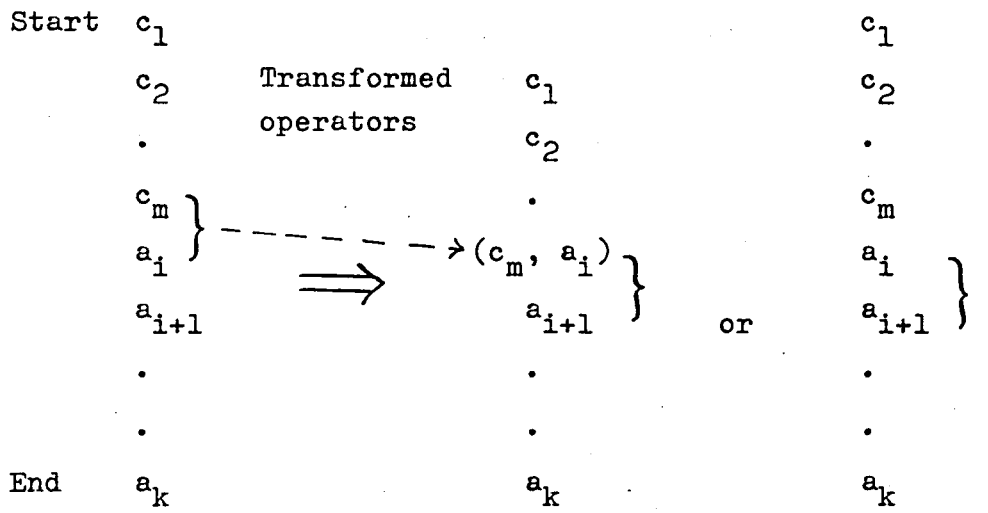
Transformation of the first step

Fig. 4.5

(ii) In general at the i-th step, the transformation of the program is as shown in Fig. 4.6. The number of steps m depends on the transformation thus far. ($1 \leq m < i$).

We apply the rule for the pair, c_m and a_i, where c_m is the concurrent compound operator and a_i is the simple operator. If there is a formula, which is applicable to this pair, rewrite them by the right hand operator, say, $c'_m = (c_m, a_i)$. If not,

leave c_m as it is, and try the next pair, a_i, a_{i+1}.



Transformation of the i-th step

Fig. 4.6

(iii) The procedure ends when a_k is transformed.

It is seen from the rule that if each pair is transformed, the result of transformation of a branch is only one concurrent compound operator.

§ 4.4.3 Transformation formulas

In the formula $\overline{X}/Y$ stands for the set of the registers, which are contained by $\overline{X}$ but are not Y.

Formulas are divided into two classes. Type A formula is applied to the pair of simple operators and type B formula to the pair of the concurrent compound

operator and the simple operator. The former is included by the latter, but it is explicitely presented for the sake of easy understanding.

The formula, for which the symbol * is attached, causes the compound operation. Such a formula can not be applied, if the compound operation can not be implemented because of, say, the distortion of the wave forms in the logical elements. If the number of the concurrent operations is restricted, those formulas, which produce the new concurrent operation, can not be applied.

Transformation formulas of type A

(1)* $\left.\begin{matrix}\alpha_1(AC,X)\rightarrow AC\\ \alpha_2(AC,Y)\rightarrow AC\end{matrix}\right\} \Rightarrow \alpha_2(\alpha_1(AC,X),Y)\rightarrow AC$

(2)* $\left.\begin{matrix}\alpha(AC,X)\rightarrow AC\\ \beta(AC)\rightarrow AC\end{matrix}\right\} \Rightarrow \beta(\alpha(AC,X))\rightarrow AC$

(3)* $\left.\begin{matrix}\beta(AC)\rightarrow AC\\ \alpha(AC,X)\rightarrow AC\end{matrix}\right\} \Rightarrow \alpha(\beta(AC),X)\rightarrow AC$

(4) $\left.\begin{matrix}\alpha(AC,X)\rightarrow AC\\ \bullet(AC)\rightarrow Y\end{matrix}\right\} \Rightarrow \alpha(AC,X)\rightarrow\{AC,Y\}$

(5) $\left.\begin{matrix}\alpha(AC,X)\rightarrow AC\\ \bullet(Y)\rightarrow AC\end{matrix}\right\} \Rightarrow \bullet(Y)\rightarrow AC$

(6) $\left.\begin{matrix}\bullet(AC)\rightarrow X\\ \alpha(AC,Y)\rightarrow AC\end{matrix}\right\}$ $\Rightarrow \bullet(AC)\rightarrow X \cap\alpha(AC,Y)\rightarrow AC,$ if $X \neq Y$

$\Rightarrow \bullet(AC)\rightarrow X \cap\alpha(AC,AC)\rightarrow AC,$ if $X = Y$

(7) $\left.\begin{array}{l} \bullet(X) \rightarrow AC \\ \alpha(AC,Y) \rightarrow AC \end{array}\right\} \Rightarrow \alpha(X,Y) \rightarrow AC$

(8)* $\left.\begin{array}{l} \beta_1(AC) \rightarrow AC \\ \beta_2(AC) \rightarrow AC \end{array}\right\} \Rightarrow \beta_2(\beta_1(AC)) \rightarrow AC$

(9) $\left.\begin{array}{l} \beta(AC) \rightarrow AC \\ \bullet(AC) \rightarrow X \end{array}\right\} \Rightarrow \beta(AC) \rightarrow \{AC, X\}$

(10) $\left.\begin{array}{l} \bullet(AC) \rightarrow X \\ \beta(AC) \rightarrow AC \end{array}\right\} \Rightarrow \bullet(AC) \rightarrow X \cap \beta(AC) \rightarrow AC$

(11) $\left.\begin{array}{l} \beta(AC) \rightarrow AC \\ \bullet(X) \rightarrow AC \end{array}\right\} \Rightarrow \bullet(X) \rightarrow AC$

(12) $\left.\begin{array}{l} \bullet(X) \rightarrow AC \\ \beta(AC) \rightarrow AC \end{array}\right\} \Rightarrow \beta(X) \rightarrow AC$

(13) $\left.\begin{array}{l} \bullet(AC) \rightarrow X \\ \bullet(AC) \rightarrow Y \end{array}\right\} \Rightarrow \bullet(AC) \rightarrow \{X, Y\}$

(14) $\left.\begin{array}{l} \bullet(AC) \rightarrow X \\ \bullet(Y) \rightarrow AC \end{array}\right\} \Rightarrow \begin{array}{ll} \bullet(AC) \rightarrow X \cap \bullet(Y) \rightarrow AC & \text{if } X \neq Y \\ \bullet(AC) \rightarrow X & \text{if } X = Y \end{array}$

(15) $\left.\begin{array}{l} \bullet(Y) \rightarrow AC \\ \bullet(AC) \rightarrow X \end{array}\right\} \Rightarrow \begin{array}{ll} \bullet(Y) \rightarrow \{AC, X\} & \text{if } X \neq Y \\ \bullet(X) \rightarrow AC & \text{if } X = Y \end{array}$

(16) $\left.\begin{array}{l} \bullet(X) \rightarrow AC \\ \bullet(Y) \rightarrow AC \end{array}\right\} \Rightarrow \bullet(Y) \rightarrow AC$

Transformation formulas of type B

(1)* $\left.\begin{array}{l} \bigcap_i \alpha_i(\overline{X}_i) \rightarrow \overline{Z}_i \\ \alpha(AC,X) \rightarrow AC \end{array}\right\} \Rightarrow \bigcap_{i \neq j} \alpha_i(\overline{X}_i) \rightarrow \overline{Z}_i \cap \alpha(\alpha_j(\overline{X}_j), X) \rightarrow AC$

$\cap \alpha_j(\overline{X}_j) \rightarrow \overline{Z}_j / AC$ if $AC \in \overline{Z}_j$ and $X \notin \cup \overline{Z}_i$

$$\Rightarrow \bigcap_i \alpha_i(\overline{X}_i)\to\overline{Z}_i \cap \alpha(AC, \alpha_j(\overline{X}_j))\to AC$$

if $X \in \overline{Z}_j$ and $AC \notin \cup\overline{Z}_i$

$$\Rightarrow \bigcap_{i\neq j} \alpha_i(\overline{X}_i)\to\overline{Z}_i \cap \alpha_j(\overline{X}_j)\to\overline{Z}_j/AC$$

$$\cap\alpha(\alpha_j(\overline{X}_j), \alpha_k(\overline{X}_k))\to AC$$

if $AC \in \overline{Z}_j$ and $X \in \overline{Z}_k$

$$\Rightarrow \bigcap_i \alpha_i(\overline{X}_i)\to\overline{Z}_i \cap \alpha(AC,X)\to AC$$

otherwise

(2)* $\left.\begin{array}{l}\bigcap_i \alpha_i(\overline{X}_i)\to\overline{Z}_i \\ \beta(AC)\to AC\end{array}\right\} \Rightarrow \bigcap_{i\neq j} \alpha_i(\overline{X}_i)\to\overline{Z}_i \cap \alpha_j(\overline{X}_i)\to\overline{Z}_j/AC$

$$\cap\beta(\alpha_j(\overline{X}_j))\to AC \quad \text{if } AC \in \overline{Z}_j$$

$$\Rightarrow \bigcap_i \alpha_i(\overline{X}_i)\to\overline{Z}_i \cap \beta(AC)\to AC \quad \text{if not}$$

(3) $\left.\begin{array}{l}\bigcap_i \alpha_i(\overline{X}_i)\to\overline{Z}_i \\ \bullet(AC)\to X\end{array}\right\} \Rightarrow \bigcap_{i\neq j} \alpha_i(\overline{X}_i)\to\overline{Z}_i \cap \alpha_j(\overline{X}_j)\to\overline{Z}_j/X$

$$\cap\bullet(AC)\to X \quad \text{if } X \in \overline{Z}_j \text{ and } AC \notin \cup\overline{Z}_i$$

$$\Rightarrow \bigcap_i \alpha_i(\overline{X}_i)\to\overline{Z}_i \cap \alpha_j(\overline{X}_j)\to X$$

if $AC \in \overline{Z}_j$ and $X \notin \cup\overline{Z}_i$

$$\Rightarrow \bigcap_{i\neq k} \alpha_i(\overline{X}_i)\to\overline{Z}_i \cap \alpha_i(\overline{X}_j)\to X$$

$$\cap\alpha_k(\overline{X}_k)\to\overline{Z}_k/X$$

if $AC \in \overline{Z}_j$ and $X \in \overline{Z}_k$

$$\Rightarrow \bigcap_i \alpha_i(\overline{X}_i)\to\overline{Z}_i \cap \bullet(AC)\to X \quad \text{otherwise}$$

(4) $\left.\begin{array}{l}\bigcap_i \alpha_i(\bar{X}_i)\to\bar{Z}_i \\ \bullet(X)\to AC\end{array}\right\} \Rightarrow \bigcap_{i\neq j}\alpha_i(\bar{X}_i)\to\bar{Z}_i \cap \alpha_j(\bar{X}_j)\to\bar{Z}_j/AC$

$\cap \bullet(X)\to AC$ if $AC\in\bar{Z}_j$ and $X\notin\cup\bar{Z}_i$

$\Rightarrow \bigcap_{i\neq k}\alpha_i(\bar{X}_i)\to\bar{Z}_i \cap \alpha_j(\bar{X}_j)\to AC$

if $X\in\bar{Z}_j$ and $AC\notin\cup\bar{Z}_i$

$\Rightarrow \bigcap_{i\neq k}\alpha_i(\bar{X}_i)\to\bar{Z}_i \cap \alpha_k(\bar{X}_k)\to\bar{Z}_k/AC$

$\cap \alpha_j(\bar{X}_j)\to AC$ if $AC\in\bar{Z}_k$ and $X\in\bar{Z}_j$

$\Rightarrow \bigcap_i \alpha_i(\bar{X}_i)\to\bar{Z}_i \cap \bullet(X)\to AC$ otherwise

Transformation formula for conditional jump

If the conditional jump is the last operator of the branch, the following formula is applied.

$\left.\begin{array}{l}\bigcap_i \alpha_i(\bar{X}_i)\to\bar{Z}_i \\ \xi_{AC}(X,SC)\to SC\end{array}\right\} \Rightarrow \bigcap_i \alpha_i(\bar{X}_i)\to\bar{Z}_i \cap \xi_{AC}(X,SC)\to SC$

if $AC\in\cup\bar{Z}_i$

$\bigcap_i \alpha_i(\bar{X}_i)\to\bar{Z}_i \cap \xi_F(X,SC)\to SC$

otherwise,

where F means the logical function, which computes the new value of AC.

§ 4.4.4 Post-edit of results

After applying the transformation formulas to every branches and connecting branches to reconstruct the program sequence according to the sequence diagram, we obtain the object program. The object program is not, however, the most efficient one and needs "post-edit" by man. The post-edit is made in order to brush up the object program and to make it more practical. The definite rule of post-edit has not been reached but the program is processed case by case with a few exceptions of the general rule.

Consider the following situation. There may happen to occur the non-sense register operation such as $\bullet(X) \longrightarrow \bullet(X)$. This is obtained from the transformation of the pair (DRW X, STO X). This operation has no effect and should be eliminated. Furthermore, if X is not used by other operations, the register X itself can be eliminated. In general the following rule is applicable:

Starting at the operator, where some register symbol, say X, appears at the right hand of the register operation for the first time, and tracing all sequences following it, if X is not found at the left hand of other operators, then the register symbol X can be eliminated.

By this rule two registers were eliminated in the case of TAKE WEIGHT.

The other technique to eliminate the symbol and reduce the program steps is to change the name of register. This technique is, however, applied by intuition of the human being and can not be formulated completely. The registers, which are eliminated by post-edit, were used for the intermediate memory in the source program because our KM has only one AC.

The logical circuits indicated in the next section for example are those which were obtained by translation and by post-edit after that.

§ 4.5 Examples of transformation

The program TAKE WEIGHT of Fig. 3.4A is transformed for example. Consider the branch, which starts at c. The procefure of transformation is indicated in Fig. 4.7.

c	$\bullet(AC)\rightarrow A$ $\bullet(C)\rightarrow AC$ }	$\bullet(AC)\rightarrow A \cap \bullet(C)\rightarrow AC$
	$\bullet(AC)\rightarrow B$	$\bullet(AC)\rightarrow A \cap \bullet(C)\rightarrow \{AC,B\}$
	$EOR(A,AC)\rightarrow AC$	$EOR(C,AC)\rightarrow AC \cap \bullet(AC)\rightarrow A \cap \bullet(C)\rightarrow B$
	$\bullet(AC)\rightarrow C$	$EOR(C,AC)\rightarrow \{AC,C\} \cap \bullet(AC)\rightarrow A \cap \bullet(C)\rightarrow B$
	$\bullet(B)\rightarrow AC$	$EOR(C,AC)\rightarrow C \cap \bullet(AC)\rightarrow A \cap \bullet(C)\rightarrow \{B,AC\}$
	$AND(AC,A)\rightarrow AC$	$EOR(C,AC)\rightarrow C \cap AND(C,AC)\rightarrow AC \cap \bullet(AC)\rightarrow A \cap \bullet(C)\rightarrow B$

Example of transformation of a branch

Fig. 4.7

Thus this branch is transformed to the single concurrent compound operator,

c EOR(AC,C)→C⋂AND(C,AC)→AC⋂•(AC)→A⋂•(C)→B.

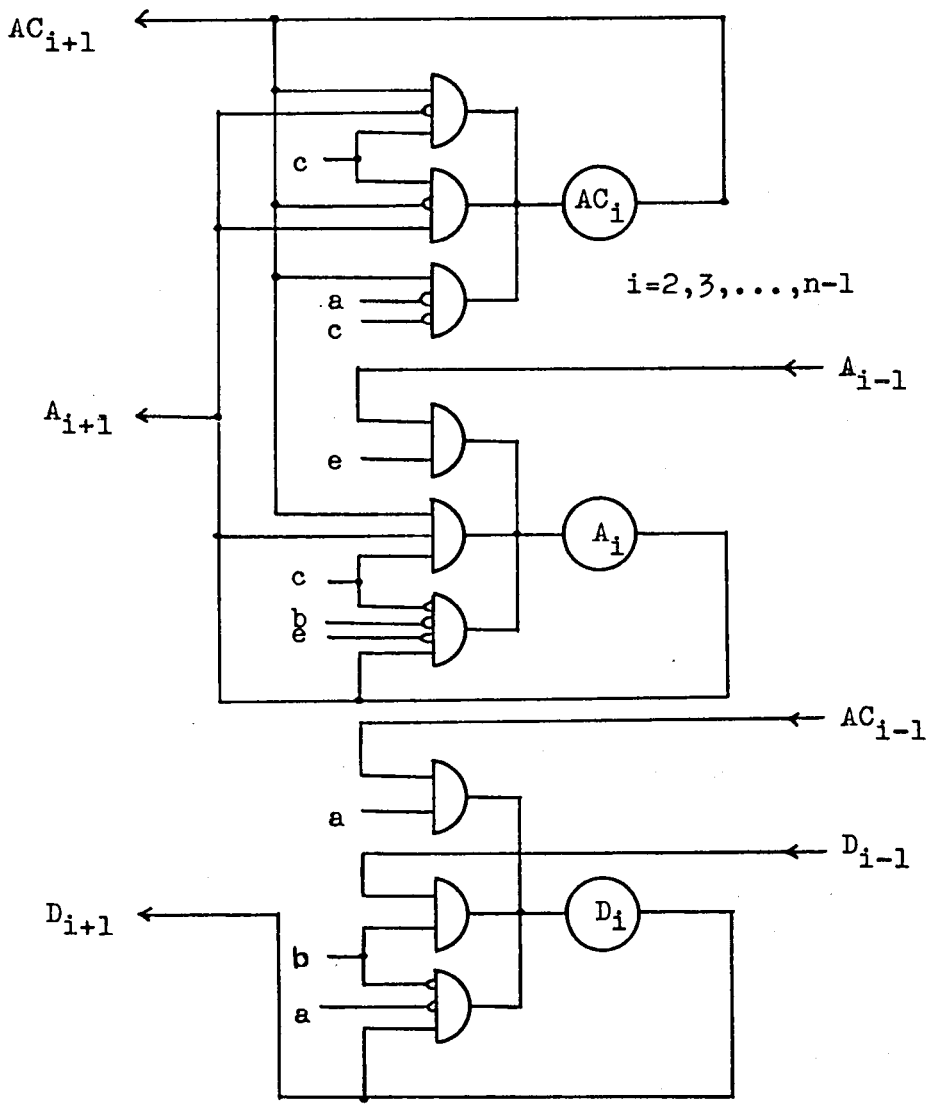
The whole object program, which was post-edited, is shown in Fig. 4.8, where the formula for the conditional jump is not applied.

START	$JNZ_{AC}(a,SC) \rightarrow SC$
HALT	HALT
a	$LSH1(AC) \rightarrow D \cap RSH\overline{n-1}(AC) \rightarrow AC$
d	$JNZ_{D}(b,SC) \rightarrow SC$
	JMP HALT
b	$LSH1(D) \rightarrow D \cap RSH\overline{n-1}(D) \rightarrow A$
	$JNZ_{A}(c,SC) \rightarrow SC$
	JMP d
c	$EOR(A,AC) \rightarrow AC \cap AND(A,AC) \rightarrow A$
	$JNZ_{A}(e,SC) \rightarrow SC$
	JMP d
e	$LSH1(A) \rightarrow A$
	JMP c

Object program of TAKE WEIGHT

Fig. 4.8

The logical circuit of the i-th bit, which corresponds to the object program of Fig. 4.8 is shown in Fig. 4.9. Its timing circuit is given in Fig. 4.10.



Operation part of TAKE WEIGHT

Fig. 4.9

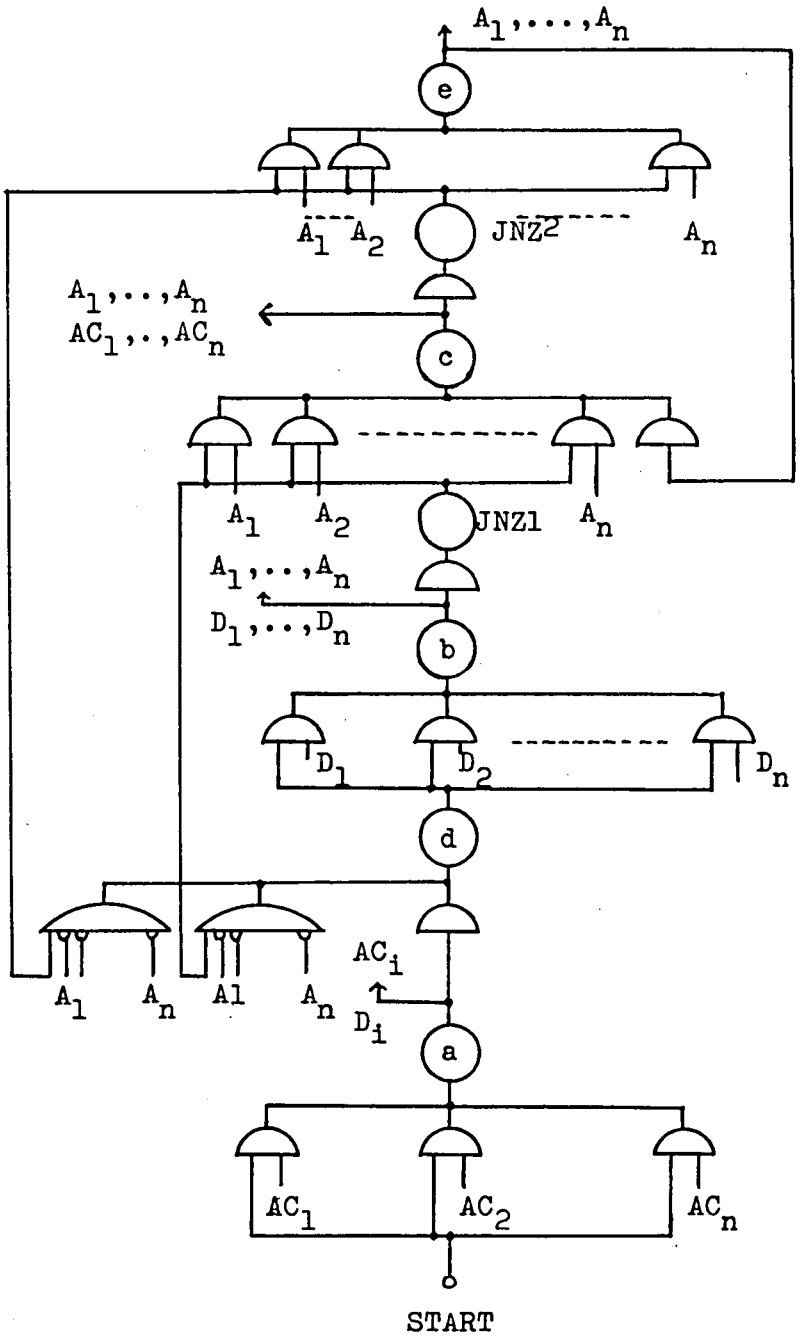
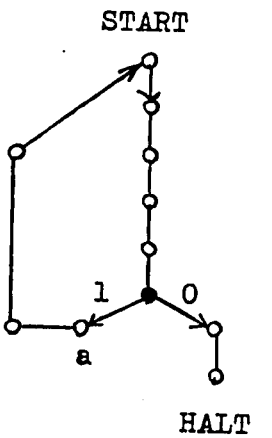


Fig.4.10 Timing circuit of TAKE WEIGHT

The program "ADD X", which is addition of the contents of AC and memory register X in the binary number system, is indicated in Fig. 4.11. The object program, which is obtained by transformation and post-edit, is shown in Fig. 4.12. Among other tdchniques, the names of the registers have been changed by post-edit. By this technique two registers were eliminated. The object circuit corresponding to Fig. 4.12 is shown in Fig. 4.13.

```
    "ADD   X"
START  STO   B
       EOR   X
       STO   C
       DRW   B
       AND   X
       JNZ   a
       DRW   C
HALT   HALT
  a    LSH   1
       STO   X
       DRW   C
       JMP   START
```



Source program of "ADD X"

Fig. 4.11

```
START  •(X) ⟶ A
  b    EOR(AC,A) ⟶ AC  ∩  AND(AC,A) ⟶ A
       JNZ_A(a,SC) ⟶ SC
HALT   HALT
  a    LSH1(A) ⟶ A
       JMP   b
```

Fig. 4.12 Object program of "ADD X"

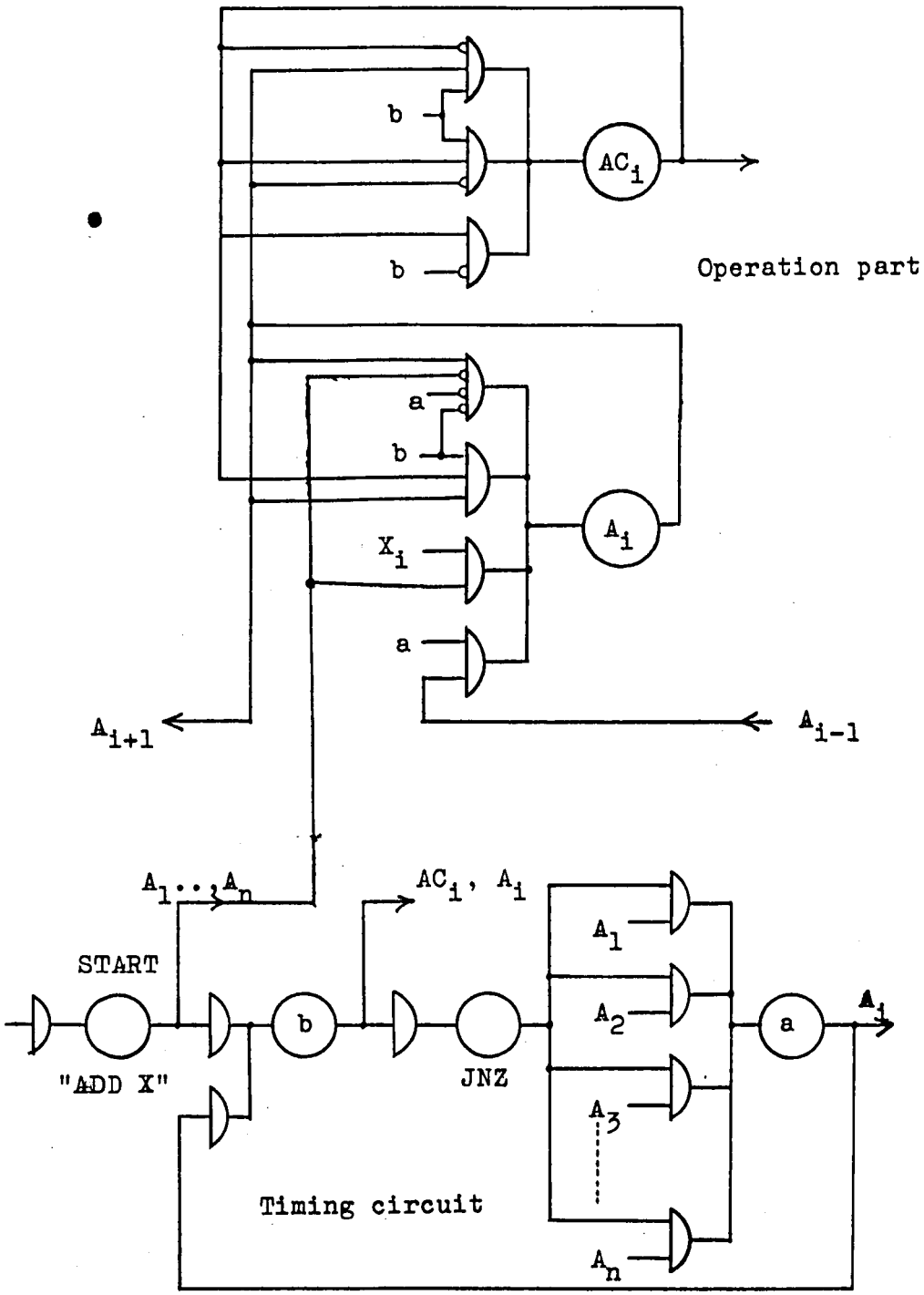


Fig. 4.13 Adder designed by translation

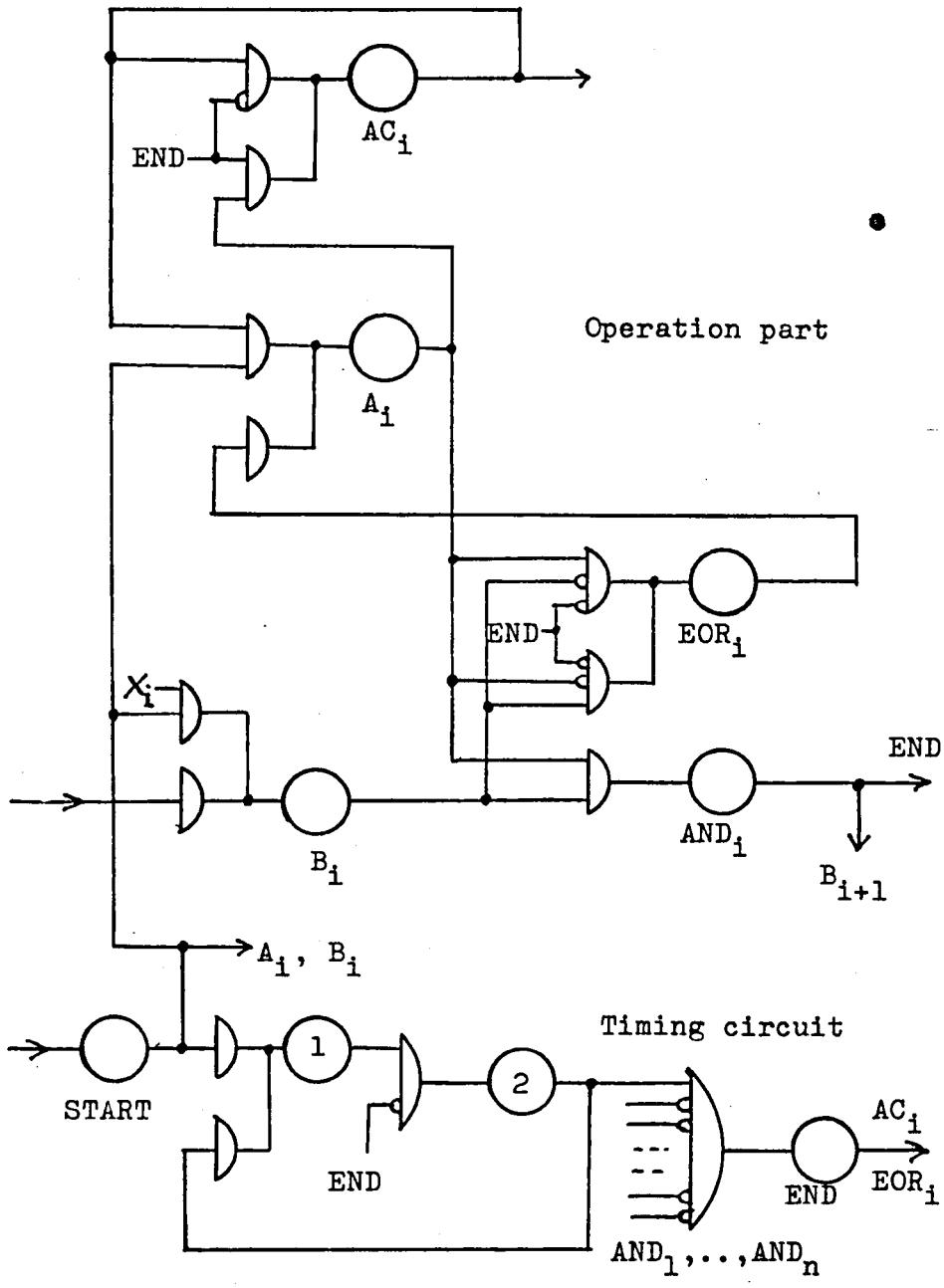


Fig. 4.14 Adder designed by man

Fig. 4.14 shows the adder, which was designed by a student independently from our translation. It is shown for the purpose of comparison. The timing circuits are the same for both circuits. The operation part of this method uses much elements than that of our method.

We investigated some other programs. Among them there are MULT, which is the multiplication of two binary numbers and CMP, which compares the absolute values of two binary numbers and branches the program sequence according to the comparison. These examples suggest some interesting problems. Above all the dependence of the object machine on the difference of the source program for the same job is interesting. It seems that the operation part may be affected by programming on the different principle, but the structure of loops of the timing circuit remains unaltered.

LITERATURE

PART II

[1] M.V.Wilkes and J.B.Stringer (1953) Micro-programming and the design of the control circuits of an electronic digital computer: Proc. Cambridge Phil. Soc. 49 230-238

[2] A.W.Burks and H.Wang (1957) The logic of automata PART I and PART II: J.A.C.M. 193-218, 279-297

[3] D.F.Gorman and J.P.Anderson (1962) A logic design translator: Proceedings of 1962 Fall J.C.C.

[4] R.W.Allard, K.A.Wolf and R.A.Zemlin (1964) Some effects of the 6600 computer on language structures: Communications of the ACM, 7 No.2 112-118

[5] R.M.Proctor (1964) A logic design translator experiment demonstrating relationships of language to systems and logic design: IEEE EC-13

[6] H.P.Schlaeppi (1964) A formal language for describing machine logic timing, and sequencing (LOTIS): IEEE EC-13 439-448

[7] J.Nievergelt (1965) On the automatic simplification of computer programs: Communication of the ACM 8 No.6 366-370

[8] A.A.Markov (1954) Theory of algorithms: Academy of Sciences of the USSR. Works of the Math.Inst. jmeni V.A. steklov 42

[9] T.C.Bartee, I.L.Lebow and I.S.Reed (1962) Theory and design of digital machines: MacGraw Hill 324pp

[10] Hidenosuke Nishio (1961) Über die Berechenbarkeit der Mikro-Programmierung: Master thesis Dept. Elect. Eng. Kyoto Univ.

[11] Toshiyuki Sakai and Hidenosuke Nishio (1965) Translation of program into logical circuit: IECEJ* Electronic computer technical committee 64-1-14

坂井利之、西尾英之助 (1965) プログラムから論理回路への変換: 電気通信学会 電子計算機研究会資料 64-1-14

IECEJ* The Institute of Electrical Communication Engineers of Japan.